UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMATICA

TESIS DE GRADO

“Elaboración de métricas para evaluar la seguridad del software durante su desarrollo”

Postulante: Humberto Benjamín Huanca Aliaga

Revisor: Lic. Javier Reyes Pacheco

Tutor: Lic. Grover A. Rodríguez

La Paz - Bolivia

DEDICATORIA

A Dios.

Por permitirme llegar a este momento tan especial en mi vida. Por los triunfos y los momentos difíciles que me han enseñado a valorarte cada día más, y porque sé que junto con mi querido abuelito Pedro a quien también dedico este trabajo, cuidan y guían mi camino.

A mis padres Abdon y Adela.

Por su comprensión y ayuda en momentos malos y menos malos. Me han enseñado a encarar las adversidades sin perder nunca la dignidad ni desfallecer en el intento. Me han dado todo lo que soy como persona, mis valores, mis principios, mi perseverancia y mi empeño, y todo ello con una gran dosis de amor y sin pedir nada a cambio.

A mis hermanos Reynaldo y María.

A mis hermano Reynaldo, porque siempre fue un apoyo y guía en los momentos de obscuridad y siempre he contado con él para todo, gracias a la confianza que nos hemos tenido.

A mi hermanita María René porque con su alegría, ternura y cariño fue, es y será siempre un aliciente para no caer en el largo camino de la vida.

A todos, espero no defraudarlos y contar siempre con su valioso apoyo, sincero e incondicional.

AGRADECIMIENTOS

A la conclusión de un trabajo tan arduo y lleno de dificultades es inevitable que te asalte un muy humano egocentrismo que te lleva a concentrar la mayor parte del mérito en el aporte que has hecho. Sin embargo, el análisis objetivo te muestra inmediatamente que la magnitud de ese aporte hubiese sido imposible sin la participación de personas que han facilitado las cosas apara que este trabajo llegue a feliz término. Por ello, es para mí un verdadero placer utilizar este espacio para ser justo y consecuente con ellas, expresándoles mis agradecimientos.

Deseo expresar mi gratitud a la carrera de Informática de la Universidad Mayor de San Andrés por haberme instruido y darme las herramientas y los medios para formarme profesionalmente.

Debo agradecer de manera especial al Lic. Grover A. Rodríguez por aceptarme para realizar esta tesis bajo su dirección. Su apoyo y confianza en mi trabajo y su capacidad para guiar mis ideas ha sido un aporte invaluable, no solamente en el desarrollo de esta tesis, sino también en mi formación como profesional. Las ideas propias, siempre enmarcadas en su orientación y rigurosidad, han sido la clave del buen trabajo que hemos realizado juntos, el cual no se puede concebir sin su siempre oportuna participación. Le agradezco también el haberme facilitado siempre los medios suficientes para llevar a cabo todas las actividades propuestas durante el desarrollo de esta tesis. Muchas gracias Licenciado.

.

Quiero expresar también mi más sincero agradecimiento al Lic. Javier Reyes Pacheco por su importante aporte y participación activa en el desarrollo de esta tesis. Debo destacar, por encima de todo, su disponibilidad y paciencia que hizo que nuestras discusiones redundaran benéficamente tanto a nivel científico como personal. No cabe duda que su participación ha enriquecido el trabajo realizado y, además, ha significado el surgimiento de una sólida amistad.

A todo el plantel docente y administrativo de la carrera y en especial a los funcionarios de la biblioteca de informática por su tolerancia hacia mi persona.

Por último quiero agradecer a mis compañeros y a ese grupo de amigos entrañables, amigos del alma, amigos de toda la vida, condescendientes, que lo aguantan todo, a cambio muchas veces de nada, con los que compartí inolvidables momentos de alegría y tristeza a lo largo de mi carrera.

A todas las personas que de alguna forma me han colaborado gracias muchas gracias.

RESUMEN

Podemos considerar a la seguridad del software como la seguridad de la información comúnmente conocida, la cual incluye temas como: la protección de los sistemas de información en contra del acceso no autorizado y la modificación de información. El objetivo del ciclo de vida de desarrollo de un producto software y las métricas asociadas al mismo, es desarrollar un producto que posea la calidad necesaria y suficiente para que satisfaga las necesidades y requerimientos del cliente. En nuestro trabajo presentamos un marco conceptual de calidad y proponemos un modelo, métodos, procedimientos, criterios y herramientas a emplear para medir la seguridad de un producto software y posteriormente realizar las correspondientes conclusiones y recomendaciones que nos sirvan para la retroalimentación y mejora del producto software. Además se establecerán los medios necesarios a emplear en el proceso de definición y medición de las características y atributos principales que van a formar el modelo de calidad.

INDICE

INDICE DE TABLAS Y FIGURAS

PRESENTACION

1.1 INTRODUCCION

El término calidad en la ingeniería del software hace referencia a un conjunto de cualidades que caracterizan y determinan la utilidad de un producto software, algunos parámetros que reúne el concepto de calidad son: eficiencia, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, integridad y seguridad. Para verificar que un producto software se desarrolló o se está desarrollando correctamente, se realizan pruebas y aplican métricas relacionadas con algunos de los parámetros de calidad mencionados, puesto que las métricas del software proporcionan un parámetro objetivo para su evaluación.

Algunos parámetros de calidad pueden ser medidos y varían de un sistema a otro, por ejemplo un producto software que va a ser explotado por 5 o más años, necesita ser confiable, mantenible y flexible para disminuir los costos de mantenimiento y perfeccionamiento durante el tiempo de explotación. Los diferentes parámetros de calidad del producto software pueden ser evaluados una vez concluido el producto, pero esto puede resultar muy costoso si se detectan problemas derivados de imperfecciones en el diseño o en el análisis, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida de desarrollo del software.

Para las empresas, la información es un elemento de gran importancia, puesto que ayuda en la toma de decisiones; este objetivo será logrado si la información es rápida, veraz, oportuna, eficiente y confiable; en los productos software relacionados con el área financiera, de salud o de gestión académica, la información es confidencial y personal, por lo que es necesario implementar mecanismos de seguridad en el software para proteger la información adecuadamente ante los posibles riesgos; así entonces es necesario contar con un software seguro, es decir, con aplicaciones que cumplan una serie de características, por ejemplo: que den acceso a la información que utilizan solo los usuarios autorizados (autenticándolos cuando sea necesario), que protejan los

datos frente a la manipulación inconsciente de los usuarios y que su disponibilidad esté garantizada, asimismo implica pensar en la seguridad del software desde el primer momento y desde las primeras etapas en el ciclo de vida de éste, conociendo y analizando las amenazas que pueden afectar al software que se está desarrollando.

En el presente trabajo se realizará el estudio orientado a la identificación de aspectos relevantes en el ciclo de desarrollo del software, relacionados con la seguridad a fin de proponer métricas que permitan la evaluación del logro de este criterio de calidad en algunas de las etapas del ciclo de desarrollo del software

1.2 PROBLEMÁTICA

1.2.1 PLANTEAMIENTO DEL PROBLEMA

Cuando se planifica un proyecto, se tiene que obtener estimaciones del costo y esfuerzo requerido por medio de mediciones que se utilizan para recolectar los datos cualitativos y cuantitativos acerca del software y sus procesos para aumentar su calidad

Los fallos de seguridad del software en: centros de cómputo, sistemas de archivos, bases de datos y aplicaciones, están en la mira de personas mal intencionadas, las cuales de acuerdo a su habilidad pueden ser más o menos peligrosas con el propósito de obtener información relevante de una organización; la seguridad técnica de los sistemas es un requisito indispensable; pero más allá de la seguridad técnica los administradores del sistema necesitan tener confianza en que el sistema de información permitirá alcanzar los objetivos propuestos. En este contexto, las métricas aparecen como necesarias para conocer el estado de la seguridad durante las etapas de desarrollo del software.

En principio, podría parecer que la necesidad de medir es algo evidente, después de todo es lo que nos permite cuantificar y por consiguiente gestionar de forma más efectiva algunos parámetros de calidad. Pero la realidad puede ser muy diferente, pues la medición conlleva una gran controversia y discusión en torno a la producción de software, porque en el desarrollo de software influye un gran número de variables, por lo que resulta complicado ponerse de acuerdo sobre qué medir y cómo se va a evaluar esas medidas; es muy difícil obtener resultados globales, aunque podríamos determinar bajo qué circunstancias una opción es mejor que otra.

1.3 PREGUNTA DE INVESTIGACION

- ¿Es posible expresar la seguridad del software en términos numéricos durante el ciclo de desarrollo?

1.4 OBJETIVOS

1.4.1 OBJETIVO GENERAL

Elaborar métricas que permitan evaluar la seguridad proporcionada por el software durante el desarrollo del mismo.

1.4.2 OBJETIVOS ESPECIFICOS

- Identificar las vulnerabilidades de un producto software.
- Establecer los mecanismos de prevención ante las vulnerabilidades identificadas.
- Elaborar un estudio sobre los diferentes métodos actuales de medición de la calidad del software.

- Analizar los riesgos del software, soluciones y su relación con el desarrollo del mismo.

1.5 JUSTIFICACION

1.5.1 JUSTIFICACION SOCIAL

Las grandes empresas en la actualidad se hacen más competitivas dentro de su ramo y cada vez adoptan nuevas estrategias a fin de garantizar el éxito. Actualmente en la era donde la información es automatizada y además es parte esencial de toda organización, es necesario tomar las medidas adecuadas para proteger dicha información.

La necesidad de plantear métricas para evaluar la seguridad en el proceso de desarrollo del software, incidirá en la optimización del mismo, ya que mediante su análisis se podrán establecer los lineamientos a seguir en cuanto al desarrollo del software, requeridos para incrementar el nivel de seguridad del mismo.

Por lo tanto, genera beneficios expresados en la optimización de la seguridad que repercutirá en la calidad del producto software y en la satisfacción de los usuarios finales.

1.5.2 JUSTIFICACION CIENTIFICA

El presente trabajo generará discusión sobre el conocimiento existente del área investigada y aportará nuevas herramientas en el campo de la ingeniería del software, puesto que las métricas técnicas del software proporcionan una manera sistemática para evaluarlo. Las métricas de seguridad propuestas en este trabajo proporcionarán al ingeniero de software información sobre las medidas que se

están tomando en cuenta y sobre el nivel de seguridad en el que se encuentra el software en una determinada etapa de su desarrollo.

1.6 HIPOTESIS

Los atributos cuantificables en el desarrollo de un producto software ayudan a determinar el nivel de seguridad que ofrece el mismo.

MARCO TEORICO

2.1 CALIDAD DEL SOFTWARE

2.1.1 ANTECEDENTES

A lo largo de toda la historia la búsqueda y el afán de perfección por parte del hombre ha sido constante, de tal forma, que el interés por el trabajo bien hecho y la necesidad de asumir responsabilidades sobre la labor efectuada poco a poco derivó en el concepto de calidad.

Con la revolución industrial comienza a desaparecer el artesanado, se crean grandes organizaciones y los antiguos artesanos se transforman en los trabajadores de las empresas. En esta época Taylor elaboró su teoría acerca de la "gestión científica del trabajo", cuyo objeto fue la preparación de normas para que los trabajadores las cumpliesen. Comenzó con ello la instauración paulatina de la división del trabajo, lo que suponía que los operarios interviniesen solamente en algunas operaciones del proceso productivo. Este hecho provocó la necesidad de que surgiese la figura de los empleados dedicados a tareas de inspección, aunque se prestaba más atención a la forma de realizar el trabajo (los procesos) que a la calidad de los productos.

Finalmente el control de calidad moderno o control de calidad estadístico comenzó en los años 30 del siglo XX con la aplicación industrial del cuadro de control ideado por el Dr. W. A. Shewhart, de Bell Laboratories, que fue el inventor de los conocidos gráficos de control.

Continuando en este proceso cronológico destaca el hecho de que pasados unos años del final de la II Guerra Mundial los japoneses comienzan a hacer verdadero énfasis en la calidad. En 1950 la Unión de Científicos e Ingenieros Japoneses realizó un seminario cuyo conferenciante, el Dr. W.E. Deming, desarrolló los siguientes temas: Cómo mejorar la calidad mediante el ciclo de planear, hacer,

verificar y actuar. La importancia de captar la dispersión en las estadísticas. Control de procesos mediante el empleo de cuadros de control y su aplicación. [1]

2.1.2 CONCEPTO DE CALIDAD

"Conjunto de esfuerzos efectivos de los diferentes grupos de una organización para la integración del desarrollo, del mantenimiento y de la superación de la calidad de un producto, con el fin de hacer posible la fabricación y servicio a satisfacción completa del consumidor y al nivel más económico"
[Feigenbaun, Deming y Juran]

"La mejor calidad que una empresa puede producir con su tecnología de producción y capacidades de proceso actuales, y que satisfará las necesidades de los clientes, en función de factores tales como el coste y el uso previsto" [Dr. Kaoru Ishikawa]

"La gestión de calidad en la empresa es el proceso de identificar, aceptar, satisfacer y superar constantemente las expectativas y necesidades de todos los colectivos humanos relacionados con ella, clientes, empleados, directivos, propietarios, proveedores y la comunidad con respecto a los productos y servicios que esta proporciona" [consultora Arthur Andersen]

De todas estas definiciones se extraen una serie de parámetros básicos que definen la calidad: si se desea producir productos y servicios de buena calidad para el consumidor será necesario decidir por adelantado que calidad de producto (o servicio) planificar (calidad de diseño), producir (calidad de fabricación) y vender (calidad que desea el cliente).

El American Heritage Dictionary define calidad como "una característica o atributo de algo". Como atributo de un elemento, la calidad se refiere a características mensurables, es decir: cosas que se pueden comparar para conocer estándares.

como longitud, color propiedades eléctricas y maleabilidad. Sin embargo, el software, principalmente una unidad intelectual, es más difícil de caracterizar que los objetos físicos.

2.1.3 CALIDAD DEL SOFTWARE

A la hora de definir la calidad del software se debe diferenciar entre la calidad del producto software y la calidad del proceso de desarrollo de éste (calidad de diseño y fabricación). No obstante, las metas que se establezcan para la calidad del producto van a determinar los objetivos a establecer de calidad del proceso de desarrollo, ya que la calidad del primero va a depender, entre otros aspectos, de ésta. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto. Este proceso constituye el objeto del presente trabajo.

Pero la calidad del producto software se diferencia de la calidad de otros productos de fabricación industrial, ya que el software tiene sus propias características específicas:

- El software es un producto mental, no restringido por las leyes de la Física o por los límites de los procesos de fabricación. Es algo abstracto, es algo intangible.

- Se desarrolla, no se fabrica. El coste está fundamentalmente en el proceso de diseño, no en la posterior producción en serie, y los errores se introducen también en el diseño, no en la producción.

- Los costes del desarrollo de software se concentran en las tareas de Ingeniería, mientras que en la fabricación clásica los costes se acentúan más en las tareas de producción.

- El software no se deteriora con el tiempo. No es susceptible de los efectos del entorno y su curva de fallos es muy diferente de la del hardware. Todos los problemas que surjan durante el mantenimiento estaban allí desde el principio y afectan a todas las copias del mismo, no se generan nuevos errores.

- Es artesanal en gran medida. El software, en su mayoría, se construye a medida, en vez de ser construido ensamblando componentes existentes y ya probados, lo que dificulta aún más el control de su calidad.

- El mantenimiento del software es mucho más complejo que el mantenimiento del hardware. Cuando un componente del hardware se deteriora se sustituye por una pieza de repuesto, pero cada fallo en el software implica un error en el diseño o en el proceso mediante el cual se tradujo el diseño en código máquina ejecutable.

- Es engañosamente fácil realizar cambios sobre un producto software, pero los efectos de estos cambios se pueden propagar de forma explosiva e incontrolada.

- Como disciplina, el desarrollo de software es aún muy joven, por lo que las técnicas de las que dispone aún no están perfeccionadas.

- El software con errores no se rechaza. Se asume que es inevitable que el software presente algunos errores de poca importancia.

También es importante destacar que la calidad de un producto software debe ser considerada en todos sus estados de evolución (especificaciones, diseño, códigos,...). No basta con verificar la calidad del producto una vez finalizado cuando los problemas de mala calidad ya no tienen solución o su reparación es muy costosa. La problemática general a la que se enfrenta el software es:

- Aumento constante del tamaño y complejidad de los programas.

- Carácter dinámico e iterativo a lo largo de su ciclo de vida, es decir que los programas de software a lo largo de su vida cambian o evolucionan de una versión a otra para mejorar las prestaciones con respecto a las anteriores.

- Dificultad de conseguir productos totalmente depurados, ya que en ningún caso un programa será perfecto.

- Se dedican elevados recursos monetarios a su mantenimiento, debido a la dificultad que los proyectos de software entrañan y a la no normalización a la hora de realizar los proyectos.

- No suelen estar terminados en los plazos previstos, ni con los costes estipulados, ni cumpliendo los niveles deseables de los requisitos especificados por el usuario.

- Incrementos constantes de los costes de desarrollo debido entre otros, a unos niveles de productividad bajos.

- Los clientes tienen una alta dependencia de sus proveedores por ser en muchos casos aplicaciones a "medida".

- Procesos artesanales de producción con escasez de herramientas.

- Insuficientes procedimientos normalizados para estipular y evaluar la productividad, costes, y calidad.

No obstante, existen las mediciones de las caracteristicas de un programa. Dichas propiedades incluyen complejidad ciclomática, cohesión, número de puntos de función, lineas de código y muchas más.

2.1.4 GESTION DE CALIDAD DEL SOFTWARE

La Gestión de la Calidad de Software es un conjunto de actividades de la función general de la Dirección que determina la calidad, los objetivos y las responsabilidades. Se basa en la determinación y aplicación de las politicas de calidad de la empresa. La gestión o administración de la calidad se aplica normalmente a nivel empresa o dentro de la gestión de cada proyecto. El propósito de la Gestión de la calidad del Software es entender las expectativas del cliente en términos de calidad, y poner en práctica un plan proactivo para satisfacer esas expectativas.

Desde el punto de vista de la calidad, la Gestión de la Calidad del Software (CS) está formada por 4 partes, las cuales son: (1) Planificación de la CS, (2) Control de la CS, (3) Aseguramiento de la CS y (4) Mejora de la CS.

2.1.4.1 PLANIFICACION DE LA CALIDAD DEL SOFTWARE

Es la parte de la Gestión de la Calidad encargada de realizar el proceso administrativo de desarrollar y mantener una relación entre los objetivos y recursos de la organización; y las oportunidades cambiantes del mercado. El objetivo es modelar y remodelar los negocios y productos de la empresa, de manera que se combinen para producir un desarrollo y utilidades satisfactorias.

Los aspectos a considerar en la Planificación de la CS son: Modelos/Estándares de CS a utilizar, Costos de la CS, Recursos humanos y materiales necesarios, etc. El plan de calidad define los atributos de calidad más importantes del producto a ser desarrollado y define el proceso de evaluación de la calidad. En la

Planificación de la CS se debe determinar: (1) Rol de la Planificación, (2) Requerimientos de la CS, (3) Preparación de un Plan de CS, (4) Implementación de un Plan de CS y (5) Preparar un Manual de Calidad.

2.1.4.2 CONTROL DE LA CALIDAD DEL SOFTWARE

Son las técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en 2 objetivos fundamentales: (1) mantener bajo control un proceso y (2) eliminar las causas de los defectos en las diferentes fases del ciclo de vida. Está formado por actividades que permiten evaluar la calidad de los productos de software desarrollados. El aspecto a considerar en el Control de la CS es la "Prueba del Software".

La prueba es el proceso de ejecutar un programa con intención de encontrar defectos. Es un proceso destructivo que determina el diseño de los casos de prueba y la asignación de responsabilidades. La prueba exitosa es aquella que descubre defectos. El "caso de prueba bueno" es aquel que tiene alta probabilidad de detectar un defecto aún no descubierto. El "caso de prueba exitoso" es aquel que detecta un defecto aún no descubierto.

La prueba no es: demostración que no hay errores, demostración que el software desempeña correctamente sus funciones y establecimiento de confianza que un programa hace lo que debe hacer.

La prueba del sistema verifica que cada elemento se ajusta de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se ha integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

Después que el software se ha integrado, se dirigen un conjunto de pruebas de alto nivel. Se deben comprobar los criterios de validación establecidos durante el análisis de requisitos. La prueba de validación proporciona una visión optima de que el software satisface todos los requisitos funcionales, de comportamiento y de rendimiento. Durante la validación se usan exclusivamente técnicas de prueba de caja negra. El software, una vez validado, se debe combinar con otros elementos del sistema.

Cuando se construye un software a medida para un cliente, se llevan a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requisitos. Estas pruebas las realiza el usuario final en lugar del responsable del desarrollo de sistema. Una prueba de aceptación puede ir desde un informal paso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas.

El diseño de casos de prueba para el software o para otros productos de ingeniería puede requerir tanto esfuerzo como el propio diseño inicial del producto. Sin embargo, los Ingenieros de Software tratan las pruebas como algo sin importancia, desarrollando casos de prueba que "parezcan adecuados", pero que tienen poca garantía de ser completos. Se deben diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número de errores con la mínima cantidad de esfuerzo y tiempo posible. Cualquier producto de ingeniería puede probarse de una de estas 2 formas: prueba de caja negra y prueba de caja blanca.

Cuando se considera el software de computadora, *la prueba de caja negra* se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

La prueba de caja blanca del software se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado. Para este tipo de prueba, se deben definir todos los caminos lógicos y desarrollar casos de prueba que ejerciten la lógica del programa.

2.1.4.3 ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE (SQA)

La función de aseguramiento de la calidad tiene como finalidad primaria el determinar si las necesidades de los usuarios están siendo satisfechas adecuadamente. Otra de sus funciones, aunque no se tocará mucho en la presente investigación, es la de determinar los costos que puede causar el añadir ciertas características al producto, ya que tarde o temprano, la economía resulta ser un factor decisivo para obtener un producto de calidad. Para determinar si las necesidades de los usuarios están siendo satisfechas, se deben evaluar tres áreas:

Objetivos: Los objetivos de la organización son primero, luego vienen los requerimientos del usuario. Los objetivos de cualquier usuario deben de estar en armonía con los objetivos de la organización,

Métodos: Deben de utilizarse métodos que contengan u observen las políticas, procedimientos y estándares de la organización,

Ejecución: Optimización del uso de hardware y software al implementar los productos de software.

2.1.4.4 MEJORA DE LA CALIDAD DEL SOFTWARE

Es la parte de la Gestión de la Calidad que contribuye, por medio de las mediciones, a los análisis de los datos y auditorias, a efectuar mejoras en la calidad del software

Una Auditoria de Calidad tiene como objetivo mostrar la situación real para aportar confianza y destacar las áreas que pueden afectar adversamente esa confianza. Otro objetivo consiste en suministrar una evaluación objetiva de los productos y procesos para corroborar la conformidad con los estándares, las guías, las especificaciones y los procedimientos.

Los resultados de la auditoria son documentados y remitidos al director de la organización auditada, a la entidad auditora, y cualquier organización externa identificada en el plan de auditoria. El informe incluye la lista de elementos no conformes u otros aspectos para las posteriores revisiones y acciones. Cuando se realiza el plan de auditoria, las recomendaciones son informadas e incluidas en los resultados de la auditoria.

La calidad ha dejado de ser un tópico y es necesario que forme parte de los productos o servicios que comercializamos para nuestros clientes. El cliente es el mejor auditor de la calidad, él exige el nivel que está dispuesto a pagar por ella, pero no más. Por tanto, debemos de cuantificar cuál es el nivel de calidad que nos exige para poder planificar la calidad de los productos que se generen a lo largo de la producción del producto o servicio final. Al analizar las necesidades de nuestros clientes, deberemos tener en cuenta la previsible evolución de sus necesidades y tendencias en cuanto a características. Deberemos tener en cuenta la evolución tecnológica del entorno de producción de nuestros productos para suministrarlos con el nivel tecnológico adecuado.

2.1.5 METRICAS DE CALIDAD

Para evaluar las áreas expuestas con anterioridad, es necesario que se cuente con un programa de aseguramiento de calidad que sea efectivo y que tenga un impacto dentro del desarrollo y prueba del producto de software final. [Web01]

Además de estas actividades, el grupo de SQA coordina el control y la gestión de cambios; ayuda a recopilar y analizar las métricas del software. Las métricas son escalas de unidades sobre las cuales puede medirse un atributo cuantificable. Cuando se habla de software nos referimos a la disciplina de recopilar y analizar datos basándonos en mediciones reales de software, así como a las escalas de medición. Los atributos son características observables del producto o del proceso de software, que proporciona alguna información útil sobre el estado del producto o sobre el progreso del proyecto. El término producto se utiliza para referirse a las especificaciones, a los diseños y a los listados del código. Los valores de las métricas no se obtienen sólo por mediciones. Algunos valores de métricas se derivan de los requisitos del cliente o de los usuarios y, por lo tanto, actúan como restricciones dentro del proyecto.

Las medidas de Calidad del Software deben comenzar desde la especificación y terminar con la implementación, implantación y mantenimiento o post-implantación. Debe aplicarse a lo largo de todo el proceso de Ingeniería de Software. Básicamente, la medición es una fase normal de cualquier actividad industrial Sin mediciones es imposible perseguir objetivos comerciales normales de una manera racional.

Existen métricas a nivel Proyecto, Proceso y Producto respectivamente. Las métricas a recabar dependen de los objetivos del negocio en particular. Los desarrolladores tienen a la vez objetivos comunes como, respetar el presupuesto y respetar los plazos, minimizar las tasas de defectos antes y después de la entrega del producto e intentar mejorar la calidad y la productividad. Las métricas deben

ayudar a la evaluación de las representaciones del modelo lógico y físico, deben tener la capacidad de intuir sobre la complejidad del diseño y construcción; y deben ayudar en el diseño de casos de prueba

2.1.6 CERTIFICACION DE CALIDAD

La certificación y calificación de servicios de software constituyen un diferencial muy importante para las empresas informáticas en el contexto de fuerte competencia local y mundial. Para lograr este reconocimiento los productos software deben cumplir con ciertas normas establecidas, las cuales describiremos a continuación:

2.1.6.1 ISO 9126

Es un estándar internacional para la evaluación del Software. El estándar está dividido en cuatro partes las cuales dirigen, respectivamente, lo siguiente: modelo de calidad, métricas externas, métricas internas y calidad en las métricas de uso.
El modelo de calidad establecido en la primera parte del estándar, ISO 9126-1, clasifica la calidad del software en un conjunto estructurado de características y sub-características de la siguiente manera:

- Funcionalidad - Un conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen las necesidades implícitas o explícitas.
    - Idoneidad
    - Exactitud
    - Interoperabilidad
    - Seguridad
    - Cumplimiento de normas

- Fiabilidad - Un conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un periodo establecido.

    - Madurez
    - Recuperabilidad
    - Tolerancia a fallos

- Usabilidad - Un conjuntos de atributos relacionados con el esfuerzo necesitado para el uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.

    - Aprendizaje
    - Comprensión
    - Operatividad
    - Atractividad

- Eficiencia - Conjunto de atributos relacionados con la relación entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas

    - Comportamiento en el tiempo
    - Comportamiento de recursos

- Mantenibilidad - Conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software

    - Estabilidad
    - Facilidad de análisis
    - Facilidad de cambio
    - Facilidad de pruebas

- Portabilidad - Conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra.
    - Capacidad de instalación
    - Capacidad de reemplazamiento
    - Adaptabilidad
    - Co-Existencia

Cada sub-característica (como adaptabilidad) está dividida en atributos. Un atributo es una entidad la cual puede ser verificada o medida en el producto software. Los atributos no están definidos en el estándar, ya que varían entre diferentes productos software.

Un producto software está definido en un sentido amplio como: los ejecutables, código fuente, descripciones de arquitectura, y así. Como resultado, la noción de usuario se amplía tanto a operadores como a programadores, los cuales son usuarios de componentes como son bibliotecas software.

El estándar provee un entorno para que las organizaciones definan un modelo de calidad para el producto software. Haciendo esto así, sin embargo, se lleva a cada organización la tarea de especificar precisamente su propio modelo. Esto podría ser hecho, por ejemplo, especificando los objetivos para las métricas de calidad las cuales evalúan el grado de presencia de los atributos de calidad.

Métricas internas, son aquellas que no dependen de la ejecución del software (medidas estáticas).

Métricas externas, son aquellas aplicables al software en ejecución.

La calidad en las métricas de uso están sólo disponibles cuando el producto final es usado en condiciones reales. Idealmente, la calidad interna determina la calidad externa y esta a su vez la calidad en el uso.

El modelo de calidad McCall está organizado sobre tres tipos de Características de Calidad:

- Factores (especificar): Describen la visión externa del software, como es visto por los usuarios.

- Criterios (construir): Describen la visión interna del software, como es visto por el desarrollador.

- Métricas (controlar): Se definen y se usan para proveer una escala y método para la medida.

ISO 9126 distingue entre fallo y no conformidad. Un fallo es el incumplimiento de los requisitos previos, mientras que la no conformidad es el incumplimiento de los requisitos especificados. Una distinción similar es la que se establece entre validación y verificación

2.1.6.2 MODELO DE MCCALL

El modelo de McCall organiza los factores en tres ejes o puntos de vista: operación del producto, elaboración del producto y revisión del producto, desde los cuales el usuario puede contemplar la calidad de un producto, basándose en once factores de calidad organizados en torno a los tres ejes y a su vez cada factor se desglosa en otros criterios:

Puntos De Vista O Ejes	Factor	Criterios
OPERACIÓN DEL PRODUCTO	Facilidad de uso	- Facilidad de operación: Atributos del software que determinan la facilidad de operación del software. - Facilidad de comunicación: Atributos del software que proporcionan entradas y salidas fácilmente asimilables. - Facilidad de aprendizaje: Atributos del software que facilitan la familiarización inicial del usuario con el software y la transición del modo actual de operación. - Formación: El grado en que el software ayuda para permitir que nuevos usuarios apliquen el sistema.
	Integridad	- Control de accesos. Atributos del software que proporcionan control de acceso al software y los datos que maneja. - Facilidad de auditoría: Atributos del software que facilitan la auditoría de los accesos al software. - Seguridad: La disponibilidad de mecanismos que controlen o protejan los programas o los datos.
	Corrección	- Completitud: Atributos del software que proporcionan la implementación completa de todas las funciones requeridas. - Consistencia: Atributos del software que proporcionan uniformidad en las técnicas y notaciones de diseño e implementación.

		- Trazabilidad o rastreabilidad: Atributos del software que proporcionan una traza desde los requisitos a la implementación con respecto a un entorno operativo concreto.
TRANSICION DEL PRODUCTO	Fiabilidad	- Precisión: Atributos del software que proporcionan el grado de precisión requerido en los cálculos y los resultados. - Consistencia. - Tolerancia a fallos: Atributos del software que posibilitan la continuidad del funcionamiento bajo condiciones no usuales. - Modularidad: Atributos del software que proporcionan una estructura de módulos altamente independientes. - Simplicidad: Atributos del software que posibilitan la implementación de funciones de la forma más comprensible posible. - Exactitud: La precisión de los cálculos y del control.
	Eficiencia	- Eficiencia en ejecución: Atributos del software que minimizan el tiempo de procesamiento. - Eficiencia en almacenamiento: Atributos del software que minimizan el espacio de almacenamiento necesario.
REVISION DEL PRODUCTO	Facilidad de mantenimiento	- Modularidad. - Simplicidad. - Consistencia. - Concisión: Atributos del software que posibilitan la implementación de una función con la menor cantidad de códigos posible.

		- Auto descripción: Atributos del software que proporcionan explicaciones sobre la implementación de las funciones.
	Facilidad de prueba	- Modularidad. - Simplicidad. - Auto descripción. - Instrumentación: Atributos del software que posibilitan la observación del comportamiento del software durante su ejecución para facilitar las mediciones del uso o la identificación de errores.
	Flexibilidad	- Auto descripción. - Capacidad de expansión: Atributos del software que posibilitan la expansión del software en cuanto a capacidades funcionales y datos. - Generalidad: Atributos del software que proporcionan amplitud a las funciones implementadas. - Modularidad.
	Reusabilidad	- Auto descripción. - Generalidad. - Modularidad. -Independencia entre sistema y software: Atributos del software que determinan su dependencia del entorno operativo. - Independencia del hardware: Atributos del software que determinan su dependencia del hardware.
	Interoperabilidad	- Modularidad. - Compatibilidad de comunicaciones: Atributos del

		software que posibilitan el uso de protocolos de comunicación e interfaces estándar. - Compatibilidad de datos: Atributos del software que posibilitan el uso representaciones de datos estándar. - Estandarización en los datos: El uso de estructuras de datos y de tipos estándar a lo largo de todo el programa.
	Portabilidad	- Auto descripción. - Modularidad. -Independencia entre sistema y software. - Independencia

Tabla 2.1. Factores de calidad de Mc Call
Fuente: [Pressman07]

2.2 SEGURIDAD DEL SOFTWARE

2.2.1 ¿Qué es la seguridad del software?

La seguridad del software aplica los principios de la seguridad de la información al desarrollo de software. Esto se refiere a la seguridad de información comúnmente conocida como la protección de sistemas de información contra el acceso desautorizado y la modificación de información, si está en una fase de procesamiento, almacenamiento o tránsito.

2.2.2 LA SEGURIDAD EN EL DESARROLLO DE SOFTWARE

Sin importar el tipo de software que se construya, siempre se debe considerar la seguridad como una prioridad para continuar con el desarrollo del sistema y así evitar factores externos que puedan dañar el trabajo.

En este contexto se define "política de seguridad" como el conjunto de requisitos

destinados a la protección de los recursos informáticos tanto físicos como lógicos durante la operación normal del mismo.

Las políticas de seguridad son el paso previo al despliegue de la "arquitectura de seguridad" y los "planes de prevención, contención y recuperación". Por arquitectura de seguridad se entiende el conjunto de soluciones tecnológicas destinadas a asegurar los recursos a proteger: físicos y lógicos, locales y en red; mientras que los planes de prevención hacen referencia al conjunto de normas y medidas que mantienen y regulan el nivel de seguridad en los mismos.

2.2.3 EL MITO DEL AMBIENTE HOSTIL

Son numerosos los documentos en el que se habla de ambientes hostiles, de ambientes confiables, o de entornos de bajo riesgo. La realidad es que no existen ambientes confiables. Todo ambiente confiable puede tornarse hostil, tanto en cuanto puede evolucionar ante determinadas circunstancias.

Se puede suponer que el ambiente donde uno trabaja (intranet) es confiable, un lugar donde se prueba prototipos o software en estado alfa/beta. Hasta cierto grado puede parecer un ambiente confiable y de hecho lo será, pero que puedan existir ambientes confiables en un determinado momento no significa que se pueda hacer software pensando en ambientes confiables y software pensando en ambientes hostiles. Un software debe intentar responder a unos criterios de seguridad sea cual sea el ambiente: sea una intranet, sea el propio Host local, o sea un concurso de toma de bandera en un certamen de seguridad informática. Programar con unos criterios de seguridad pobres, pensando en el escenario de destino, es algo nefasto y de consecuencias negativas la gran mayoría de las veces, ya que bastará un cambio en el entorno para que la seguridad quede comprometida.

2.2.4 VULNERABILIDAD DEL SOFTWARE

La vulnerabilidad de software puede definirse como "un fallo o hueco de seguridad detectado en algún programa o sistema informático que puede ser utilizado bien por virus para propagarse e infectar, o por hackers para entrar en los sistemas de forma no autorizada". De manera más sencilla, se trata de un fallo de diseño de alguno de los programas instalados en el ordenador, que permite que personas no autorizadas puedan realizar alguna acción maliciosa, ejecutar archivos sospechosos o que abra numerosos puertos del equipo para quien quiera introducirse en él.

Normalmente, cuando se detecta una vulnerabilidad, la compañía fabricante del software afectado publica el parche necesario para corregirla. El problema se presenta cuando algún usuario malicioso tiene conocimiento de dicho problema y rápidamente desarrolla un "exploit", o programa diseñado para aprovecharlo. Éste, entre otros usos, puede ser incorporado a un código malicioso.

2.2.5 CLASIFICACIÓN DE VULNERABILIDADES

La anatomía de una vulnerabilidad depende del contexto y área en la cual se desarrolla, el estudio de las vulnerabilidades de software tiene 5 factores básicos de investigación, las cuales dependen del contexto en el que se desarrollen, estos factores son los siguientes:

- Las fallas encontradas
- Las técnicas y métodos de descubrimiento
- El nivel de gravedad de la vulnerabilidad
- La autenticación
- Las consecuencias de la misma

2.2.6 FALLOS DE SEGURIDAD CLÁSICOS

2.2.6.1 DESBORDAMIENTO DE BÚFER (BUFFER OVERFLOW)

El búfer es un espacio reservado en memoria para el almacenamiento de datos, el desbordamiento de búfer (Buffer Overflow), es un error común en el proceso de desarrollo de software, es la sobre escritura de un área en memoria no contemplado por el programador, debido a que el espacio definido para el almacenamiento de los datos no es suficiente.

Estas áreas no contempladas por el programador, son áreas protegidas por el sistema operativo, la sobre escritura en memoria de estas, genera una violación de acceso a espacio en memoria, dando como resultado el término inesperado de la aplicación en proceso.

En la figura 2.1 se muestra una representación gráfica de un búfer en memoria definido por el programador.

Figura 2.1: Memoria definida por el programador

En la figura 2.2 se representa una sobre escritura de las áreas protegidas por el sistema operativo generando así una excepción y la terminación inesperada del programa en proceso.

Figura 2.2: Sobre escritura de los datos protegidos

En las figuras 2.1 y 2.2 se muestra simplemente una representación gráfica de lo que sucede con los datos contenidos en memoria, cada celda representada por una letra, en esencia contiene los datos almacenados en binario, para mejor comprensión y en todos los casos los datos son representados en numeración hexadecimal, cada celda en memoria, contiene un apuntador a ella, el cual es denominado puntero de memoria u Offset.

Con este ejemplo se comprende básicamente el almacenamiento de los datos en memoria y lo que puede ocasionar el desbordamiento de un búfer.

2.2.6.2 SQL INJECTION

SQL INJECTION es una vulnerabilidad informática en el nivel de la validación de las entradas a la base de datos de una aplicación. El origen es el filtrado incorrecto de las variables utilizadas en las partes del programa con código SQL. Es, de hecho, un error de una clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o de script que esté incrustado dentro de otro.

Una inyección SQL sucede cuando se inserta o "inyecta" un código SQL "invasor" dentro de otro código SQL para alterar su funcionamiento normal, y hacer que se ejecute maliciosamente el código "invasor" en la base de datos.

La inyección SQL es un problema de seguridad informática que debe ser tomado en cuenta por el programador para prevenirlo. Un programa hecho con descuido, displicencia, o con ignorancia sobre el problema, podrá ser vulnerable y la seguridad del sistema puede quedar ciertamente comprometida. Esto puede suceder tanto en programas ejecutándose en computadores de escritorio, como en páginas Web, ya que éstas pueden funcionar mediante programas ejecutándose en el servidor que las aloja.

La vulnerabilidad puede ocurrir cuando un programa "arma" descuidadamente una sentencia SQL, con parámetros dados por el usuario, para luego hacer una consulta a una base de datos. Dentro de los parámetros dados por el usuario podría venir el código SQL inyectado.

Al ejecutarse esa consulta por la base de datos, el código SQL inyectado también se ejecutará y podría hacer un sinnúmero de cosas, como insertar registros, modificar o eliminar datos, autorizar accesos e, incluso, ejecutar código malicioso en el computador.

Asumiendo que el siguiente código está en una aplicación y que existe un parámetro "nombreUsuario" que contiene el nombre de usuario que nosotros le demos, la inyección SQL es posible:

Consulta: "SELECT * FROM usuarios WHERE nombre = '" + nombreUsuario + "';"

Si el usuario escribe su nombre, digamos "Alicia", nada anormal sucedería, la aplicación generaría una sentencia SQL similar a la siguiente, que es perfectamente correcta, en donde se seleccionaría al usuario "Alicia":

SELECT * FROM usuarios WHERE nombre = 'Alicia';

Pero si un usuario malintencionado escribe como nombre de usuario:

"Alicia'; DROP TABLE usuarios; SELECT * FROM datos WHERE '-' = '-", se generaría la siguiente consulta SQL:

(El color verde es lo que pretende el programador, el azul es el dato, y el rojo, el código SQL inyectado)

```
SELECT * FROM usuarios WHERE nombre = 'Alicia';
DROP TABLE usuarios;
SELECT * FROM datos WHERE '-' = '-';
```

La base de datos ejecutaría la consulta en orden, seleccionaría el usuario 'Alicia', borraría la tabla 'usuarios' y seleccionaría datos que quizá no están disponibles para los usuarios comunes. En resumen, cualquier dato de la base de datos está disponible para ser leído o modificado por un usuario malintencionado. Nótese por qué se llama "Inyección SQL". Si observamos el código malicioso, de color rojo, vemos que está en el medio del código bueno, el verde. El código rojo ha sido "inyectado" dentro del verde.

2.2.6.3 HTML INJECTION

Consiste en hacer una inyección de código HTML en una página, las webs más vulnerables suelen ser libros de visitas, blogs, foros o cosas por el estilo. Así que es posible ingresar al buscador google y encontrar páginas vulnerables utilizando las terminaciones de las urls o contenido.

Ej:

allinurl:action=post
allinurl:book.php
allinurl:forum.php
etc., etc., etc.

La inyección es muy sencilla de hacer, lo primero que tenemos que hacer es comprobar si al administrador se le ha olvidado deshabilitar el HTML en los posts y los temas, así que crearemos un tema nuevo o agregaremos un comentario con código html para comprobar si funciona o no. Se puede utilizar lo que sea, por ejemplo:

<html>
<marquee>Probando vulnerabilidad</marquee> </html>

Se observa lo que se ha posteado y ese tag hace que las letras se muevan, si no se mueven, es que el código HTML, esta deshabilitado, así que tendremos que ir a buscar otra víctima.

Una vez encontrada una web vulnerable, solo es cuestión de imaginación para hacer un desfase pequeño, se podría agregar HTML, Javascript, FLASH; como ejemplo se puede establecer una instrucción para cerrar su página y que automáticamente se abra otra página, redirigirlos a otra dirección, hacer un bucle infinito en javascript con alertas o ventanas.

2.2.6.4 ERRORES DE FORMATO

En este epígrafe se agrupa 2 errores comunes: los errores en el formato de las cadenas en la librería libc, y el desbordamiento de los enteros.

El primer error está vinculado a la familia de funciones "printf", y plantea un problema de seguridad cuando esta familia de funciones es llamada sin especificar el formato de los parámetros que van a ser suministrados.

Uso correcto: printf("%s", valor);
Uso incorrecto: printf(valor);

Un error en el formato puede llevar a una violación de segmento. No vamos a entrar en cómo se explotan ese tipo de fallos, más allá de comentar muy por encima que están vinculados al uso malintencionado por parte de un usuario de las cadenas de formato propias de esta familia de funciones, más concretamente del modificador %n, el cual sirve para escribir el número bytes impresos.

2.2.7 EVOLUCION DEL FALLO DE SEGURIDAD

En el anterior punto hemos visto los que los errores clásicos tienen una serie de características, unas negativas desde el punto de vista de la explotación ser dependientes del lenguaje, ser dependientes de la arquitectura o posibilidades de error en la explotación elevadas y otras positivas privilegios elevados o simplemente ser la única forma de atacar escenarios concretos.

Sin embargo 2 han sido los factores que han hecho que el fallo de seguridad evolucione: el nivel de madurez en el código de los servicios de red y la proliferación de las aplicaciones web. La primera no es más que el producto lógico de la refactorización de un código probado durante largos años. En la actualidad los errores clásicos presentes en httpd's, ftpd's o smtpd's se han ido reduciendo cada vez más, son meses, o incluso años, los que estos servicios permanecen sin fallos de seguridad críticos, lo cual ha hecho que las vulnerabilidades en los sistemas evolucionen hasta nuevos ámbitos, esta evolución ha venido de la mano de aplicaciones desarrolladas por capas y concretamente de las aplicaciones web.

2.2.8 OTROS FALLOS DE SEGURIDAD

2.2.8.1 ESCALADA DE DIRECTORIOS

En cualquier tipo de aplicación es posible que existan medidas de seguridad que restrinjan los directorios en los cuales un usuario de la misma puede interactuar, sobrepasar esas medidas de seguridad, es decir, poder acceder a directorios fuera

del marco de seguridad original, es lo que se conoce como escalada de directorios

Este error se puede presentar y explotar de muchas maneras. Una forma puede ser introduciendo los caracteres "../", como parte de una entrada de usuario relativa a un fichero que queramos visualizar, descargar o almacenar. Otra forma menos común es haciendo uso de funcionalidades adicionales las cuales aun perteneciendo a la aplicación no validen dicha restricción de seguridad. Un ejemplo de este último modelo presente PHP 4/5 hasta su versión más reciente es el siguiente

Las funciones cURL en PHP4 y PHP5 sobrepasan la proteccion open_basedir, directiva que limita el path absoluto al que un usuario puede tener acceso, de tal forma que se puede navegar a través del sistema de ficheros sin ninguna limitación, más que la inherente a los permisos del mismo.

Por ejemplo, configurando "open_basedir" en php.ini a "/var/www/html" cualquiera puede leer
"/etc/parla" usando funciones de cURL.

== Demostración del concepto (curl.php)

```
<?php
$ch = curl_init("file:///etc/parla");
$file=curl_exec($ch);
echo $file
?>
```

$ links -dump http://localhost/curltest/curl.php don't read please!

La solución a este problema pasa por la existencia de una única función que implemente la funcionalidad y sea siempre usada en toda llamada, así como controlar las extensiones de la aplicación y su respeto por dicha funcionalidad.

2.2.8.2 ERRORES DE MECAMISMOS DE AUTENTICACION

Bajo este epígrafe se agrupa un problema al que no vamos a dedicar excesivas líneas, porque bien ya ha sido tratado con errores concretos: XSS, SQL Injection. O bien pertenece al campo del diseño lógico. En este último campo, podemos encontrar desde identificadores de sesión secuenciales, hasta inexistencia de mecanismos de autenticación, pasando por un amplio surtido de debilidades como limitación de longitud de la clave, uso de sistemas de recuperación de contraseñas ineficientes, por citar algunos.

La solución a estos problemas pasa por no cometer los errores vistos con anterioridad en la codificación del mismo, a la par que diseñar correctamente aquellos que implementemos.

2.2.8.3 ERRORES EN EL MECANISMO DE CIFRADO

Este será el último error que se trate bajo él agrupamos una serie de errores al implementar sistemas de cifrado. Uso de algoritmos débiles como pueden ser RC4 o DES. Almacenaje de contraseñas en texto plano y no de los hashes md5 o sha1 de las mismas, incluso sha-256/512 en caso de querer ampliar el espacio de colisiones. Almacenaje de las contraseñas de cifrado dentro de la aplicación. O uso de algoritmos de cifrado propietario cuya seguridad no ha sido contrastada.

La solución a este problema pasa por cifrar haciendo uso de algoritmos de probada calidad como blowfish o twofish para el cifrado simétrico, RSA o DH para el cifrado asimétrico y SHA1 o SHA-256 para la generación de Hashes. Almacenar siempre los hashes de las contraseñas y nunca estas. No usar algoritmos propietarios de dudosa calidad. Y sobre todo no contener el password de cifrado en la propia aplicación, el usuario debe ser el que la introduzca, no el sistema.

2.2.9 MEDIDA, MEDICION Y METRICA

Aunque medida, medición y métrica son términos que suelen utilizarse de manera intercambiable, es importante observar las sutiles diferencias entre ellos.

En el contexto de la ingeniería del software una medida proporciona un a indicación cuantitativa de la extensión, la cantidad, la dimensión o el tamaño de algún atributo de un producto o proceso. Medición es el acto de determinar una medida. El glosario de estándares del IEEE define métrica como una "medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo determinado".

En general, podemos decir que métrica es una forma de medir y una escala definida para realizar mediciones de uno o varios atributos. Un ingeniero de software recopila medidas y desarrolla métricas para obtener los indicadores. Un indicador es una métrica o una combinación de métricas que proporcionan conocimientos acerca del proceso del software, un proyecto de software o el propio producto. En resumen:

- Métrica.- Medida cuantitativa del grado en que un sistema posee un atributo dado. Incluye el método de medición.

- Medición.- Proceso por el cual se obtiene una medida.

- Medida.- Valor asignado al atributo de una entidad mediante una medición.

Las mediciones en el mundo físico pueden englobarse en dos categorías:

1. Medidas directas.- En el proceso de ingeniería se encuentran el costo, y el esfuerzo aplicado, las líneas de código producidas, velocidad de ejecución,

el tamaño de memoria y los defectos observados en un determinado periodo de tiempo.

2. Medidas indirectas.- Se encuentra la funcionalidad, calidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento, etc.

2.2.10 LAS METRICAS Y LA CALIDAD DEL SOFTWARE

El objetivo primordial de la ingeniería del software es producir un sistema, aplicación o producto de alta calidad. Para lograr este objetivo, los ingenieros de software deben emplear métodos efectivos junto con herramientas modernas dentro del contexto de un proceso maduro de desarrollo del software. Al mismo tiempo, un buen ingeniero del software y buenos administradores de la ingeniería del software deben medir si la alta calidad se va a llevar a cabo. A continuación se verá un conjunto de métricas del software que pueden emplearse a la valoración cuantitativa de la calidad del software.

- Métricas técnicas, se centran en las características del software, por ejemplo la complejidad lógica, el grado de modularidad. Mide la estructura del sistema, el cómo está hecho.

- Métricas de calidad. Proporcionan una indicación de cómo se ajusta el software a los requisitos explícitos e implícitos del cliente. Es decir cómo voy a medir para que mi sistema se adapte a los requisitos que me pide el cliente.

- Métricas de productividad, se centran en el rendimiento del proceso. Es decir que tan productivo va a ser el software que voy a diseñar.

- Métricas orientadas a la persona, proporcionan medidas e información sobre la forma en que la gente desarrolla el software. Son las medidas que voy a hacer del personal que desarrollará el software.

- Métricas orientadas al tamaño, es para saber en qué tiempo voy a terminar el software y cuantas personas voy a necesitar.

Se han propuesto cientos de métricas para el software de computadora, pero no todas proporcionan soporte práctico para el ingeniero de software. Algunas exigen mediciones demasiado complejas; otras son tan confusas que pocos profesionales podrían comprenderlas, y otras más violan las nociones intuitivas básicas de lo que es el software de calidad.

2.2.11 ELABORACION Y VALIDACION DE METRICAS

Según Roche [ROC94], los pasos para un proceso de medición efectiva son:

- Formulación, la derivación de medidas y métricas apropiadas para la representación del software que se considera.

- Recolección, el mecanismo con que se acumulan los datos necesarios para derivar las métricas formuladas.

- Análisis, el cálculo de las métricas y la aplicación de herramientas matemáticas.

- Interpretación, la evaluación de las métricas en un esfuerzo por conocer mejor la calidad de representación.

- Retroalimentación. Recomendaciones derivadas de la interpretación de las métricas del producto transmitidas al equipo de software.

Las métricas del software solo serán útiles si están caracterizadas de manera efectiva y se validan para probar su valor. Los siguientes principios [LET03] son representativos de muchos otros que podrían proponerse para caracterizar y validar las métricas:

- Una métrica debería tener propiedades matemáticas deseables. Es decir, el valor de la métrica debe estar en un rango significativo. Además, una métrica que pretende estar en una escala racional no debe contar con componentes que solo se miden en una escala ordinal.

- Cuando una métrica representa una característica de software que aumenta cuando se presentan rasgos positivos o que disminuye al encontrar rasgos indeseables, el valor de la métrica debe validarse empíricamente en una amplia variedad de contextos antes de publicarse o aplicarse.

Aunque la formulación, caracterización y validación son críticas, la recopilación y el análisis son las actividades que dirigen el proceso de medición. Roche [ROC94] sugiere las siguientes directrices para estas actividades:

- Siempre que sea posible, deben automatizarse la recopilación de datos y su análisis.

- Deben aplicarse técnicas estadísticas válidas para establecer relaciones entre los atributos internos del producto y las características de calidad externas.

- Para cada métrica deben establecerse directrices y recomendaciones para su interpretación.

Ejiogu [EJI91] define un conjunto de atributos que toda métrica efectiva del software debe abarcar. La métrica derivada y las medidas que llevan a ella deben ser: simples y calculables, empírica e intuitivamente persuasivas, consistentes y objetivas, consistentes en el uso de unidades y dimensiones, independientes del lenguaje de programación.

Aunque casi todas las métricas del software satisfacen estos atributos, algunas métricas de uso común no cumplen con una o dos de ellas.

MARCO DEMOSTRATIVO

3.1 RESUMEN

Este capítulo desarrolla la metodología propuesta para evaluar la seguridad del software durante su desarrollo, la metodología presenta diferentes fases de las cuales se analizará y se describirán métodos, procedimientos, criterios y herramientas a aplicar en dichas actividades.

3.2 SUBJETIVIDAD Y OBJETIVIDAD EN LAS TECNICAS DE EVALUACION

El término subjetividad se utiliza con dos significados, según se utilice para referirse al conocimiento o al sujeto. En la teoría del conocimiento, la subjetividad es una propiedad de las percepciones, argumentos y lenguajes basados en el punto de vista del sujeto, y por tanto influidos por los intereses y deseos particulares del sujeto. La propiedad opuesta es la objetividad, que los basa en un punto de vista intersubjetivo, no prejuiciado, verificable por diferentes sujetos. Desde el punto de vista de la sociología la subjetividad se refiere al campo de acción y representación de los sujetos siempre condicionados a circunstancias históricas, políticas, culturales, etc.

Debido a la subjetividad de los resultados obtenidos, se propone el uso de lógica difusa, la cual se basa en comprender los cuantificadores de nuestro lenguaje.

3.2.1 FUNCIONAMIENTO DE LA LOGICA DIFUSA

En la teoría de conjuntos difusos se definen también las operaciones de unión, intersección, diferencia, negación y otras operaciones sobre conjuntos en los que se basa esta lógica.

Para cada conjunto difuso, existe asociada una función de pertenencia para sus elementos, que indican en qué medida el elemento forma parte de ese conjunto

difuso. Las formas de las funciones de pertenencia más típicas son: trapezoidal, lineal y curva.

Se basa en reglas heurísticas de la forma SI (antecedente) ENTONCES (consecuente), donde el antecedente y el consecuente son también conjuntos difusos, ya sea puros o resultados de operar con ellos. Los métodos de inferencia para esta base de reglas deben simples, veloces y eficaces. Los resultados de dichos métodos son un área final, fruto de un conjunto de áreas solapadas entre sí (cada área es el resultado de una regla de inferencia). Para escoger una salida concreta a partir de tanta premisa difusa, el método más usado es el del centroide, en el que el resultado final será el centro de gravedad del área total resultante.

3.3 ENFOQUES CUANTITATIVOS PARA EVALUAR LA CALIDAD

Es importante resaltar que, frecuentemente, se utilizan dos conceptos relacionados a la categoría o tipo de evaluación, en consideración de sus estrategias, métodos y técnicas. Estamos hablando de la *evaluación cualitativa* o de la *evaluación cuantitativa*. Como indica Dujmovic, entre otros autores, los métodos y técnicas de evaluación cualitativa, se basan generalmente en una lista de características a ser analizadas para un producto o productos competitivos. La lista puede contener características de distinto tipo (relacionadas al ente a evaluar como producto, proceso o recurso), tales como características técnicas, de costo, etc. Luego del proceso de evaluación, en la etapa de análisis y conclusiones, los tomadores de decisión frecuentemente crean para cada sistema (a comparar) una lista de ventajas y desventajas. Finalmente, mediante un mecanismo intuitivo, comparando las ventajas y desventajas para cada característica de cada sistema, arriban a un ranquin final. Este enfoque es obviamente conveniente y atractivo cuando el objeto de la evaluación y el proceso de decisión es suficientemente simple. Sin embargo, en un proceso de evaluación, comparación y selección en donde intervienen, por ejemplo, muchas características y atributos para cada sistema seleccionado, y en donde se identifican distintas relaciones entre los

mismos, un enfoque como el anterior carece de las propiedades de precisión y justificaciones objetivas necesarias.

Esta dificultad puede minimizarse mediante el uso de un enfoque cuantitativo. La meta de una metodología, método o técnica cuantitativa es proveer un proceso de evaluación flexible, bien estructurado, suficientemente preciso, y basado en principios ingenieriles de manera de proveer indicadores cuantitativos elementales, parciales y globales los cuales son usados como base y justificación de las decisiones más óptimas.

3.4 MODELOS DE AGREGACION Y PUNTAJE

Al inicio de este capítulo, cuando mediante preguntas y comentarios, presentamos la problemática general para el proceso de evaluación, dijimos: Las decisiones y procedimientos fundamentales de la evaluación, comparación y ranquin, ¿están centradas en la intuición de los evaluadores, o están cimentadas en modelos de estructuración de características y atributos, y además, en modelos y métodos cuantitativos de cálculo de puntaje?

En esta sección, presentaremos dos enfoques cuantitativos de utilidad en las actividades de estructuración de atributos y determinación de puntajes. Particularmente, discutiremos modelos de puntaje aditivos y lineales versus modelos de decisión multiatributos no-lineales.

Respecto de la idea que sustenta a los modelos de puntaje aditivos y lineales es bastante simple. Para cada característica o sistema a evaluar y comparar se identifican *n* atributos necesarios, cuya preferencia o indicador elemental (IE) se debe computar. Supongamos que los valores individuales de $IE_1 \ldots IE_n$ están normalizados de manera que: $0 <= IE_i <= 1$; o, en la escala de porcentaje $0\ \% <= IE_i <= 100\%$.

En el caso que todos los atributos intervinientes sean equi-pesados, podríamos expresar el indicador o preferencia global (IG) mediante el uso de una sumatoria. Pero si los elementos no tienen la misma importancia en el modelo de estructuración, debemos introducir la idea de pesos positivos y normalizados, de manera que: $0 <= P_i <= 1$, para $i = 1 .. n$ y, $P_1 + ... + P_n = 1$. Por lo tanto, el puntaje o indicador global puede ser expresado mediante la siguiente expresión:

$IG = P_1 IE_1 + .. + P_n IE_n$ para $0 <= IE_i <= 1$

o, indicado de un modo más conversacional:

Puntaje Global = ä (componente Peso x componente Preferencia o Indicador Elemental)

Este enfoque puede ser adecuado para casos más simples, en donde la cantidad de atributos es suficientemente baja de modo que los pesos tengan mayor relevancia. Considere el lector, que si la cantidad de atributos es 100 el peso más bajo es casi irrelevante, en tanto que el componente peso en promedio es de 0,01 o 1% para cada indicador elemental. Esto le quita sensibilidad, flexibilidad y robustez al modelo, principalmente para evaluar sistemas de mediana o alta complejidad. Dujmovic en sus investigaciones [Dujmovic 96] identificó al menos siete inconvenientes en aplicar modelos de puntaje aditivos [Gilb 76]. Es importante enunciar dichas desventajas (algunas contenidas en dicho trabajo), para considerarlas dentro de la problemática en la evaluación de calidad de artefactos Web complejos, a saber:

1- Número limitado de componentes para la evaluación. Si la evaluación incluye *n* componentes entonces el peso promedio de cada componente es 100/n %. Esto limita el número de componentes a ser evaluados debido a la sensibilidad de los pesos.

2- *Malgasto en el esfuerzo de la evaluación*. El esfuerzo total de la evaluación depende de la cantidad de componentes y de la complejidad de sus relaciones. La

evaluación de características o atributos que tienen un efecto ínfimo en el resultado final implica malgastar recursos.

3- *Imposibilidad de modelar requerimientos obligatorios*. Los modelos aditivos no soportan requerimientos obligatorios dado que la ausencia de una característica necesaria no disminuye el puntaje parcial o global más que por el valor relativo del peso de la característica. Sin embargo, muchas veces esa ausencia no puede ser compensada.

4- *Imposibilidad de modelar requerimientos simultáneos*. Los modelos aditivos lineales no son apropiados para expresar relaciones de simultaneidad entre características y/o atributos. La aditividad asume que la presencia insuficiente de un atributo se puede siempre compensar por la suficiente presencia de *cualquier* otro atributo.

5- *Imposibilidad de modelar requerimientos suficientes*. Los modelos aditivos lineales no pueden modelar reemplazabilidad de atributos.

6- *Imposibilidad de modelar relaciones lógicas asimétricas*. Dichos modelos usan los pesos como el único medio de diferenciar la relativa importancia de las entradas.

Esto no es suficiente para expresar relaciones lógicas asimétricas tales como combinaciones de características obligatorias, deseables y opcionales; o, combinación de relaciones necesarias y suficientes.

7- *Desequilibrio en el esfuerzo de evaluación*. La evaluación de sistemas complejos implica un esfuerzo considerable. Un nivel alto de experticia se puede necesitar en este proceso. Por lo tanto, no es razonable producir costosas y precisas entradas a un modelo de decisión con baja sensibilidad.

Por ejemplo, resolver el problema de los requerimientos obligatorios no se puede satisfacer por medio del mero uso de una media geométrica. Por ello, el modelo de lógica de preferencia de puntajes (LSP), como una extensión del modelo aditivo y lineal, viene a resolver dichos inconvenientes, argumenta Dujmovic. Se puede

definir al modelo LSP como uno de agregación lógica de preferencias centrado en medias de potencia pesada. Permite modelar relaciones de reemplazabilidad, neutralidad, y simultaneidad entre atributos y características. Se puede utilizar operadores de preferencia lógica para modelar distintos nivel de intensidad de polarización "y/o", entre otros aspectos.

3.5 EL ENFOQUE PROPUESTO: Métrica de calidad para evaluar el software durante su desarrollo.

El enfoque propuesto es esencialmente integral, flexible y cubre la mayor parte de las actividades en el proceso de desarrollo del software.

Esta metodología pretende ser usada principalmente en la etapa de desarrollo del producto software, dado que muchas veces los desarrolladores de software no consideran aspectos que generalmente los toman como secundarios sin darle la importancia requerida a cada factor.

3.6 METODO DE DEMOSTRACION

El proceso demostrativo consiste en: a partir de unas proposiciones dadas que llamaremos premisas, obtener otra proposición que llamaremos conclusión, mediante la aplicación de las reglas lógicas.

En el presente trabajo emplearemos el método de demostración por contradicción o reducción al absurdo. Antes de introducirnos a este método necesitamos precisar los siguientes conceptos que forman parte de su estructura.

Contradicción - designamos de esta forma a toda proposición correspondiente a la conjunción entre una proposición y su negación.

Teoría contradictoria o inconsistente.- Cuando en dicha teoría es posible encontrar una contradicción. En una teoría contradictoria podemos concluir que una proposición es verdadera y falsa a la vez.

El método de demostración por reducción al absurdo se fundamenta en al condición de no contradicción para una teoría, básicamente la estrategia consiste en suponer explícitamente la negación de la proposición a demostrar, a partir de esta hipótesis se trata de generar una contradicción, esto es: que la teoría con ese supuesto es inconsistente y, en consecuencia, tal hipótesis es falsa, o lo que es equivalente, que su negación es verdadera, quedando validada la proposición inicial.

3.6.1 ESQUEMA OPERATIVO

Queremos demostrar que nuestra proposición P. es válida:

P: *Los atributos cuantificables en el desarrollo de un producto software ayudan a determinar el nivel de seguridad que ofrece el mismo.*

1. Suponemos la negación de la tesis (no P) como hipótesis auxiliar. Entonces nuestra hipótesis auxiliar (Haux = no P) es:

   Haux: *Los atributos cuantificables en el desarrollo del software* ***NO*** *ayudan a determinar el nivel de seguridad que ofrece el mismo*

2. A partir de las premisas de la teoría y de la hipótesis auxiliar se razona por el método directo, hasta concluir una contradicción.

3. Finalmente podemos concluir la validez de nuestra hipótesis inicial.

3.7 DEFINICION DE METRICAS: MÈTODO DE ALARCOS

Para la definición y validación de las métricas propuestas emplearemos el método de Alarcos, el cual es representado en el diagrama siguiente:

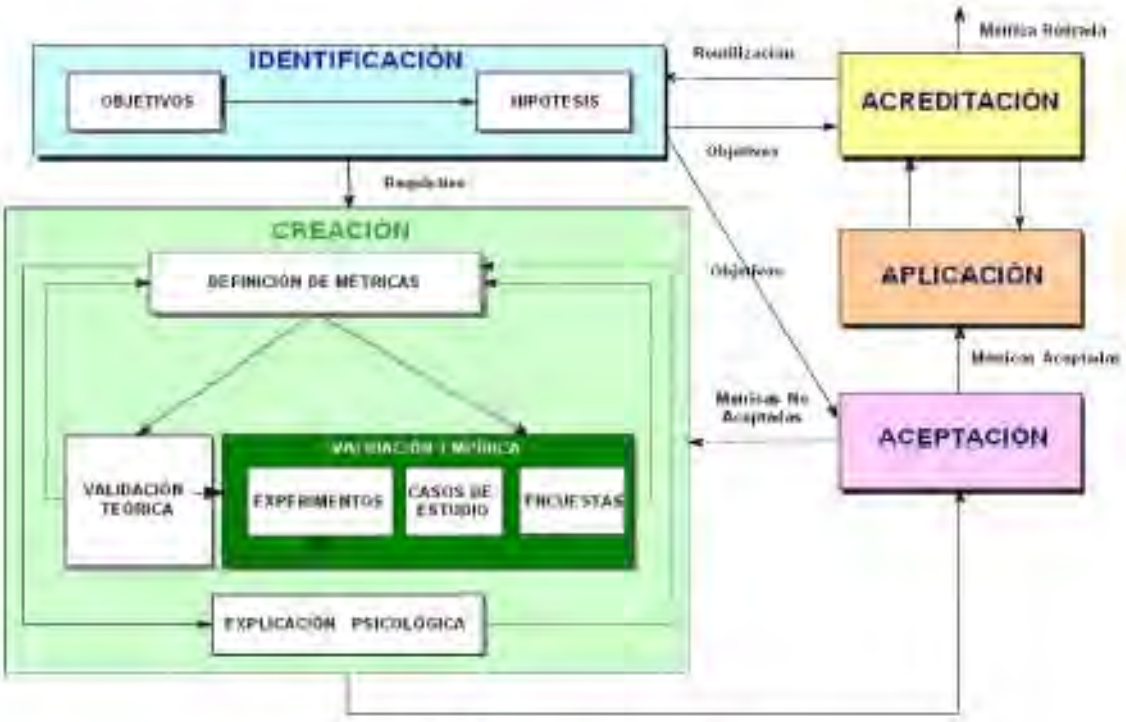

Figura 3.1: Método de Investigación de Alarcos
Fuente: Departamento de Informática
Universidad de Castilla-La Mancha

Esta metodología está compuesta de 5 etapas principales:

1. Identificación.- Se definen los objetivos que se persiguen a la hora de crear la métrica y se plantean las hipótesis de cómo se llevará a cabo la medición. Sobre los elementos de esta etapa (objetivos e hipótesis) se basarán todas las etapas siguientes. Como resultado de esta etapa se generan los requisitos que debe cumplir dicha métrica. Además, como se observa en la figura 3.0, los objetivos serán utilizados en las etapas de aceptación, aplicación y acreditación.

2. Creación - Se realiza la definición de la métrica y su validación teórica y empírica. Esta etapa es una de las más importantes y larga pues abarca un proceso iterativo del que debe salir una métrica valida tanto formal como empíricamente. Como se puede observar en la figura 3.0, el proceso de creación se subdivide en varias etapas intermedias. Como resultado de la retroalimentación, las métricas deben ser redefinidas de acuerdo a las validaciones, teóricas o empíricas, fallidas. Al final de la etapa de creación, las métricas serán consideradas válidas y aquellas que no sean validas, serán descartadas.

   a. Definición: Es el primer paso de esta fase que debe realizarse considerando las características del producto que vamos a medir y la experiencia de los profesionales. En la definición se deben considerar objetivos claros, es decir, realizar una definición de la métrica orientada al objetivo para evitar obtener una definición de la métrica que no cumple con el objetivo deseado. Es deseable que la definición de las métricas se realice de manera formal para evitar ambigüedades.

   b. Validación teórica: El objetivo principal de la validación teórica es probar si la idea intuitiva acerca del atributo que está siendo medido se refleja en la medida. Esto se hace analizando los requisitos que deben ser satisfechos cuando estamos midiendo. Además la validación teórica proporciona información relacionada con las operaciones matemáticas y estadísticas que pueden ser realizadas con la métrica, lo cual es esencial cuando tengamos que trabajar con ella.

   c. Validación empírica: El objetivo de esta etapa es probar la utilidad de las métricas propuestas. El saber general, la intuición o la especulación no son fuentes fiables de conocimiento por lo que es necesario realizar validaciones empíricas con las métricas. La

validación empírica se utiliza para obtener información objetiva sobre la utilidad de las métricas propuestas ya que puede que una métrica sea correcta desde el punto de vista formal, pero no tener relevancia práctica para un problema determinado. Así pues, el estudio empírico resulta necesario para comprobar y entender las aplicaciones de las medidas de nuestros productos. Esto se consigue a través de hipótesis en el mundo real, más allá de la pura teoría, que habrá que comprobar con datos empíricos. Kish [Kish59] divide las investigaciones empíricas en tres clases principales: experimentos, encuestas y simples investigaciones. Experimentos son las investigaciones en las que las posibles variables perturbadoras han sido aleatorizadas. Encuestas son investigaciones en las que los sujetos del estudio son una muestra representativa de la población a la que pertenecen. Investigaciones simples son aquellas en las que no hay aleatoriedad de variables perturbadoras ni representatividad de los sujetos que componen la muestra de estudio. El primer tipo corresponde con el método experimental, mientras que los otros dos se enmarcan dentro del método correlacional (casos de estudio en nuestro caso). Así pues, la forma de saber si debemos desarrollar un caso de estudio o un experimento, dependerá del grado de control que tengamos sobre las variables. Si tenemos una alto grado de control sobre las variables que pueden afectar a las hipótesis, estaremos en condiciones de realizar un experimento, mientras que si por el contrario, no va a ser posible controlar muchas de las variables afectadas, será mejor realizar un caso de estudio; en la siguiente tabla se presentan estos conceptos.

Factor	Experimentos	Casos de estudio
Nivel de control	Alto	Bajo
Dificultad de controlar	Baja	Alta
Nivel de réplica	Alto	Bajo
Coste de replicar	Bajo	Alto

Tabla 3.1. Factores para seleccionar la técnica de experimentación
Fuente: Método para la definición de métricas- Grupo Alarcos

d. Explicación psicológica. Idealmente deberíamos ser capaces de explicar la influencia de los valores de las métricas desde un punto psicológico. Algunos autores como Siau [SIAU99], proponen el uso de la psicología cognitiva como una disciplina de referencia.

3. Aceptación.- Una vez obtenida la métrica valida, suele ser necesario pasar por una etapa de aceptación de la métrica en la que se harán pruebas en entornos reales, de manera que podamos comprobar si la métrica cumple los objetivos deseados dentro del campo de la aplicación real.

4. Aplicación.- Suele ser necesaria la existencia de una fase de pruebas en laboratorio en la que se realice una experimentación sistemática en entornos reales y con usuarios reales para verificar si cumple los objetivos buscados dentro de un entorno de trabajo real. En definitiva intenta encontrar si las métricas validas que se consiguieron al final de la fase de creación son aceptables en entornos de aplicación reales, teniendo en cuenta los objetivos obtenidos en la etapa de identificación.

5. Acreditación.- Es la última etapa del proceso, que discurre en paralelo con la fase de aplicación y tiene como objetivo el mantenimiento de la métrica, de manera que se la pueda adaptar al entorno cambiante de aplicación. Como consecuencia de esta etapa, puede que una métrica sea retirada, porque ya no sea útil en el entorno en el que se aplica o que sea reutilizada para iniciar el proceso de nuevo.

3.8 PANORAMA DE LAS PRINCIPALES FASES DE LA METODOLOGIA DE EVALUACIÓN DEL SOFTWARE

En esta sección describiremos, para la metodología de evaluación del software, las principales fases, actividades, modelos, y algunos constructores intervinientes en el proceso de evaluación, comparación y ranquin de calidad. La figura 3.1 muestra una vista general de las fases de la metodología y de los principales pasos y constructores de proceso.

Estas fases son:

- *Planificación y Programación de la Evaluación de Calidad*
- *Definición y Especificación de Requerimientos de Calidad*
- *Definición e Implementación de la Evaluación Elemental*
- *Definición e Implementación de la Evaluación Global*
- *Análisis de Resultados, Conclusión y Documentación*

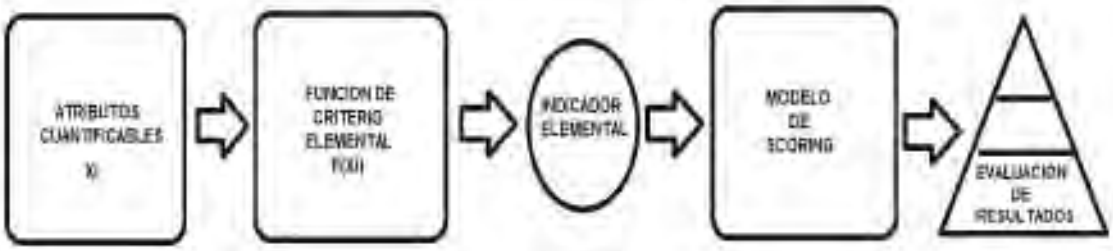

Figura 3.2: Módulos en el proceso de evaluación
Fuente ISO/14598-5

3.8.1 ATRIBUTOS CUANTIFICABLES.

Según la teoría desarrollada en el anterior capítulo, y de acuerdo al modelo establecido, para este estudio se considera el siguiente conjunto de atributos como cuantificables:

- Autenticación, referida al conjunto de parámetros relacionados con el ingreso de los usuarios al sistema, podemos identificar el número de campos de entrada, al número de intentos de la contraseña y a la complejidad que conlleva cada una de ellas.

  Para la evaluación de este atributo, se considera en principio el tamaño de la palabra clave, mientras más larga sea la cadena entre 9 y 13 caracteres se consideran mucho más seguros.

  Otro aspecto que también se toma en cuenta es el tipo de caracteres que se utiliza, por ejemplo si la cadena es solo numérica, ésta es mucho más sencilla de ser descifrada, entonces si la contraseña conlleva símbolos alfanuméricos, la clave será más segura.

- Encriptación, principalmente orientado a la seguridad de los datos almacenados en la base de datos. Según análisis y encuestas realizadas a programadores, muchos de ellos no consideran a la encriptación de datos como una forma de dotar seguridad al sistema y solo determinan que el tiempo de proceso involucrado se vería afectado, es decir que el desarrollo de los programas están en función de las características del equipo y no así en función de la seguridad de los datos.

  En este estudio se considera que el uso de algoritmos de encriptación agrega un nivel de seguridad mayor a cualquier sistema que no los tenga. Estos algoritmos pueden ser aquellos mismos otorgados por los lenguajes de programación o bien aquellos desarrollados por los mismos programadores. Es así que mientras más complejo sean los algoritmos de encriptación utilizados tanto para el almacenamiento de los datos como para su transmisión, más seguro será considerado el sistema.

- Programación. En este punto se considera la forma de programación como otro punto de debilidad en el desarrollo de los sistemas. Cuando al usuario se solicita información, como por ejemplo la introducción de una palabra clave, ésta puede ser burlada fácilmente si no se contemplan las medidas adecuadas para controlar este tipo de ataques.

  Frente a esta situación se consideran los ataques de ejecución de comando denominados *sql injection* y *html injection* como aquellos típicos ataques de los que se deben cuidar al momento de desarrollar un sistema cualquiera. De estos dos tipos de ataques, aquel que puede ser introducido como parte de una consulta sql es el que conlleva mayor peligrosidad al sistema en desarrollo.

- Abuso de funcionalidad. Por fallas de funcionalidad básicamente se comprende a aquellas fallas que se presentan por la no contemplación de recomendaciones de construcción presentadas por la arquitectura del equipo, por ejemplo los típicos errores de buffer overflow (desbordamiento), o aquellos relacionados con el manejo de las fechas; fallas que permiten a cualquier intruso realizar ataques a través de estas 'puertas abiertas' e introducir códigos maliciosos.

3.8.2 REPRESENTACION DE LAS CARACTERISTICAS Y ATRIBUTOS

A continuación, en la tabla 3.8 se presenta una categorización simple de las amenazas a un sistema, y luego un conjunto inicial de fuentes identificadas:

Categoria de amenaza	Descripción	Propiedad comprometida
Sabotaje	La ejecución del software se suspende o termina, o su funcionamiento se degrada, o el ejecutable se suprime o se destruye.	Disponibilidad
Cambio de versión	El software se modifica intencionalmente o se substituye por parte no autorizada, o se inserta en el ejecutable lógica no autorizada (código malicioso)	Integridad
Intercepción	Usuario no autorizado ingresa al sistema, o a algún modulo del mismo	Control de acceso
Descubrimiento	Se revelan aspectos tecnológicos del software y detalles de puesta en práctica empleando ingenieria reversa.	Confidencialidad

Tabla 3.2: Categorias de amenazas
Fuente: US CERT. Attack patterns

Para la asignación de los valores de ponderación que se dará a cada uno de los atributos, se realizó una encuesta (Ver Anexos) a 50 oficiales de seguridad, teniendo en consideración el objetivo de la evaluación que es realizar un estudio con el fin de valorar, determinar y comparar el estado de diferentes sistemas; además tomando en cuenta el impacto de los ya mencionados errores de seguridad, concluimos que los pesos normalizados para cada uno de los factores en estudio son:

1. Errores de diseño-autenticación	**0.2***
Control de numero de intentos	0.3
Uso de contraseñas	0.7
Longitud de Contraseñas	0.6
• 1-8 Caracteres	0.1
• 9-13 Caracteres	0.6
• 14+ Caracteres	0.9
Tipo de Contraseña	0.4
• Numérico	0.5
• Alfanumérico	0.9

2. Encriptación	**0.3***
Numero de algoritmos utilizados: • 1. . . . . . • 2 o más.	 0.7 0.9
3. Errores de programación y configuración. . . .	**0.3***
4. Abuso de funcionalidad.	**0.2***

Tabla 3.3 Ponderación de los atributos cuantificables
Fuente: Elaboración propia

3.8.3 FUNCION DE CRITERIO ELEMENTAL

De acuerdo a la tabla se distinguen los siguientes grupos claramente identificados:

Autenticación	$u_0 = \sum_{i=1}^{n} w_i a_i / \sum_{i=1}^{n} w_i$ *numero de intentos:* $a_1 = \frac{\#\ validaciones\ controladas}{\#\ validaciones\ de\ usuario}$ *uso de contraseña:* $u_2 = \sum_{i=1}^{n} w_i d_i / \sum_{i=1}^{n} w_i$ d_i = *peso segun contraseña.*

Encriptación	$u_1 = \sum_{i=0}^{n} x_i / n$ donde: $x_1 = \frac{\#\ validaciones\ protegidas}{\#\ validaciones\ de\ usuario}$ $x_2 = peso\ según\ numero\ de\ alg.$
Programación	$u_2 = \frac{\#\ consultas\ protegidas}{\#\ consultas\ a\ la\ BD}$
Abuso de funcionalidad	$u_3 = \frac{\#\ campos\ de\ entrada\ prot.}{\#\ campos\ de\ entrada}$

Tabla 3.4 Funciones de criterio elemental
Fuente: Elaboración propia

Donde

*: En la tabla 3.1 representa el peso de cada grupo.

W_i: Representa el peso de los atributos de autenticación.

n_i: representa el número de variables que pertenecen al mismo atributo.

u_i: Es la variable que asociaremos a los pesos asignados a cada uno de los grupos.

Siguiendo esta forma de razonamiento, cuando se desee ponderar los valores del primer atributo cuantificable que en este caso se refiere a los errores de autenticación, se debería proceder de la siguiente manera:

- Identificamos el nivel de seguridad de las claves que serán introducidas a la base de datos, analizando la longitud y los tipos de caracteres que contenga; de acuerdo a nuestros pesos en la tabla 3.1 y la función de ponderación asociada a cada grupo, podremos establecer un factor el cual podremos evaluarlo.

- Como cada grupo tiene un peso asociado; es decir un nivel de importancia, utilizando la media aritmética ponderada, se podrá hallar una medida centralizada de acuerdo a los resultados obtenidos en cada grupo

$$resultado\ final = \sum_{i=1}^{n} w_i x_i \Big/ \sum_{i=1}^{n} w_i$$

Esta medida de centralización resulta poco influida por la existencia de determinados valores muy grandes que el conjunto de los otros, siendo en cambio sensible a valores mucho más pequeños que el conjunto.

Al no estar definido la medida en el caso de la existencia en el conjunto de valores nulos, los valores que se debe a signar a cada una de las variables identificadas para el estudio no deben tener valores nulos, condición que se cumple al observar los pesos establecidos para cada variable.

3.8.4 MODELO DE SCORING

Ahora que se tienen cuantificados cada variable con un peso w_i se plantea ingresar estos valores a un conjunto de lógica difusa, los que determinarán en qué medida los valores introducidos tienden a pertenecer a conjuntos cuyas variables de salida son:

Valores de la variable de salida	
Calidad de desarrollo del software	• Aceptable. • Regular. • Malo.

Tabla 3.5: Valores de la variable de salida
Fuente: Elaboración Propia

El criterio de agregación tiene como entrada dos indicadores elementales a partir de sendos criterios elementales a saber el criterio elemental $CrE(X_1)$ y el $CrE(X_2)$. Esto es, para cada par de valores de las variables X_1 y X_2 se generan dos preferencias o indicadores elementales correspondientes: IE_1 e IE_2.

El problema consiste ahora en expresar la preferencia o indicador global IG_i como una función de agregación de los indicadores elementales

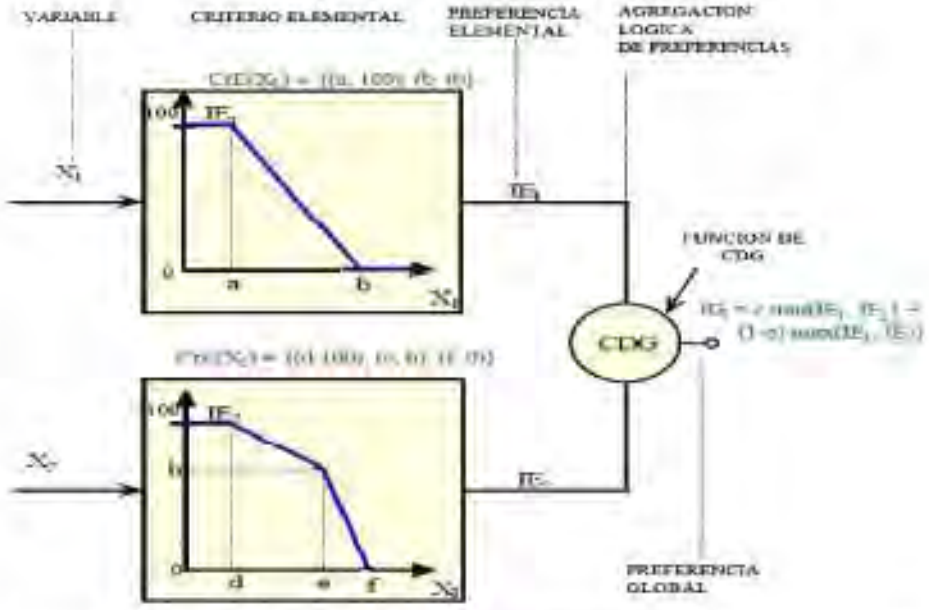

Figura 3.4: Panorama del proceso de determinación de la preferencia de calidad global a partir de preferencias elementales

Ahora determinamos la función de agregación asumiendo que IE_1 e IE_2 tienen igual importancia (o peso). Una propiedad básica de dicha función es la siguiente desigualdad:

$$\text{Min}(IE_1, IE_2) <= IG_i(IE_1, IE_2) <= \text{Max}(IE_1, IE_2) \quad (1)$$

En donde,

$\text{Min}(IE_1, IE_2) = IE_1 \dot{U} IE_2$, esto es, la conjunción;

$\text{Max}(IE_1, IE_2) = IE_1 \dot{U} IE_2$; esto es, la disyunción.

Como se estableció anteriormente, la interpretación aproximada de IE_1 es que denota el porcentaje del requerimiento elemental satisfecho conforme al valor de

X_1 y así para IE_2. Consecuentemente, la desigualdad (1) puede verbalmente ser expresada como:

a) La satisfacción global de un conjunto de requerimientos no puede ser más grande que la satisfacción del requerimiento elemental más satisfecho; y

b) La satisfacción global de un conjunto de requerimientos no puede ser menor que la satisfacción del requerimiento elemental menos satisfecho. O dicho en otras palabras: la preferencia global no puede ser más preferida (o mejor) que su parte más preferida (la mejor), ni puede ser menos preferida (o peor) que su parte menos preferida (la peor).

No obstante, los casos extremos son posibles. El caso conjuntivo, esto es, cuando $IG_i = Min\ (IE_1, IE_2)$ representa la situación en donde todos los grados de satisfacción (o preferencias elementales) pueden ser menos considerados, excepto el mínimo. Lo que significa que "*una cadena es tan resistente como lo es su eslabón más débil*"; es decir, se desea la satisfacción simultánea de todos los requerimientos. Por lo tanto, no hay modo de compensar la parte más débil de un elemento de un componente, mejorando el resto de los elementos interrelacionados del componente.

Por el contrario, el caso disyuntivo, esto es, cuando $IG_i = Max\ (IE_1, IE_2)$ representa la situación en donde todos los grados de satisfacción (o preferencias) pueden ser menos considerados, excepto el máximo. Lo que significa que el componente de un sistema es considerado tan bueno como su mejor parte (o elemento del componente). Por lo tanto, no hay modo de mejorar la preferencia global de un componente (o sistema) mejorando su parte más débil.

3.9 ESPECIFICACIÓN DEL CONJUNTO DIFUSO.

Una vez definidos las variables que deben ser consideradas para la evaluación de la métrica, se plantea desarrollar con estos valores un modelo difuso el cual determinará un valor cuantitativo. El conjunto de reglas para el siguiente modelo son especificadas en la sección de *anexos*.

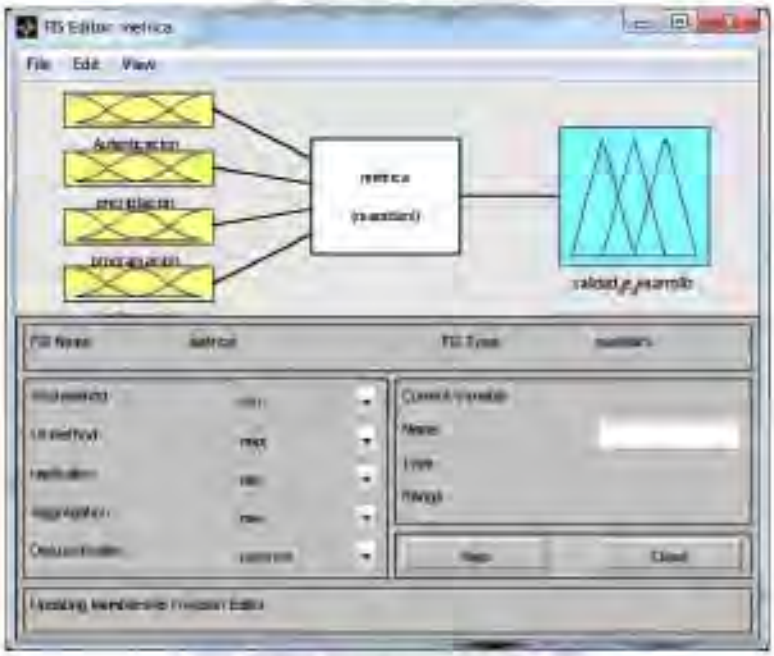

Figura 3.5 Especificación de variables
Fuente Elaboración Propia

De acuerdo al gráfico se distinguen las variables de entrada establecidos: Autenticación, encriptación, programación y el factor de configuración.

A su vez cada una de las variables difusas que se tienen, están definidas según las funciones matemáticas.

Variable de entrada:	Autenticación
Función característica:	Trapezoidal
Universo de discurso	[0..1]
Tipo de Variable:	Dependiente
Valores lingüísticos:	3

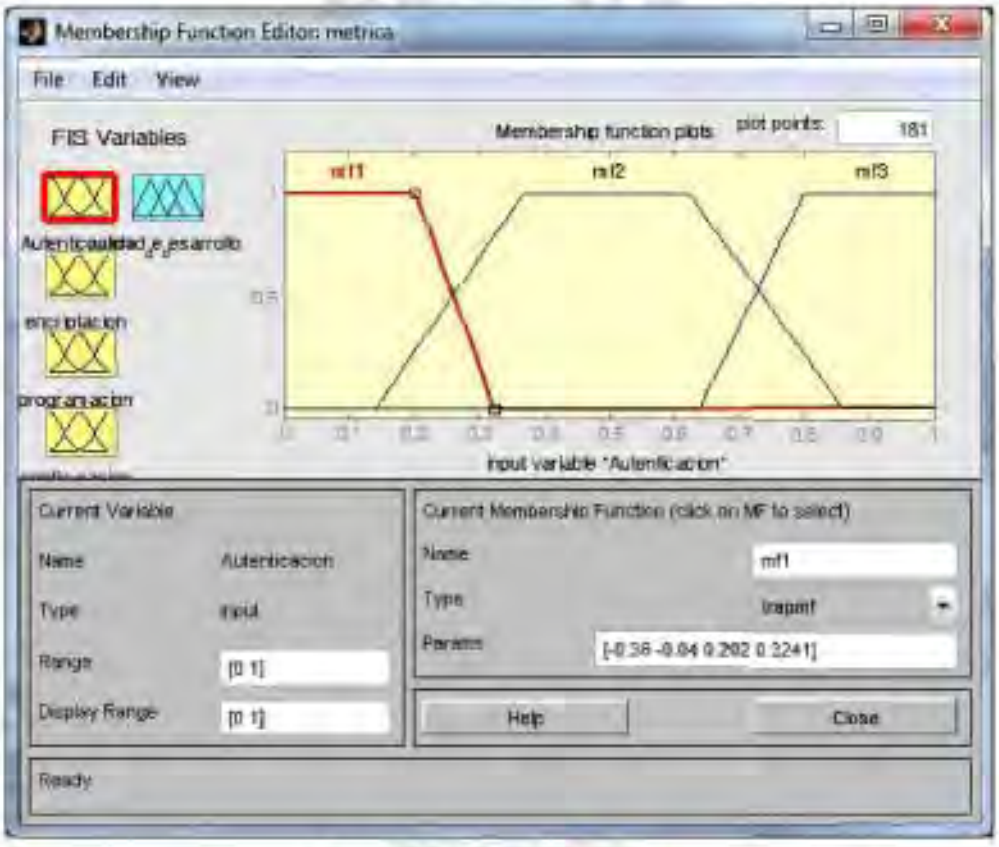

Figura 3.6 Variable autenticación
Fuente Elaboración Propia

VALOR LINGUISTICO	NOMBRE DEL CONJUNTO	FUNCION CARACTERISTICA	DEFINICION
1	BAJA	TRAPMF	$\mu(x) = \begin{cases} 0; 0 > x \\ \square \\ 1; 0 \leq x \leq 0{,}2 \\ \square \\ \frac{0{,}35 - x}{0{,}15}; 0{,}2 < x \leq 0{,}35 \\ \square \\ 0; 0.35 < x \end{cases}$
2	MEDIA	TRAPMF	$\mu(x) = \begin{cases} 0; 0 \leq x < 0.15 \\ \square \\ \frac{0.35 - x}{0.2}; 0.15 \leq x < 0.35 \\ \square \\ 1; 0.35 \leq x \leq 0.65 \\ \square \\ \frac{0.65 - x}{0.2}; 0.65 < x \leq 0.85 \\ \square \\ 0; 0.85 < x \end{cases}$
3	ALTA	TRAPMF	$\mu(x) = \begin{cases} 0; 0.65 > x \\ \square \\ \frac{0.65 - x}{0.1}; 0.65 \leq x \leq 0.75 \\ \square \\ 1; 0.75 < x \leq 1 \\ \square \\ 0; 1 < x \end{cases}$

Tabla 3.6: Función autenticación
Fuente: Elaboración Propia

Variables de entrada:	Encriptación
Función característica:	Trapezoidal
Universo de discurso	[0..1]
Tipo de Variable:	Dependiente
Valores lingüísticos:	3

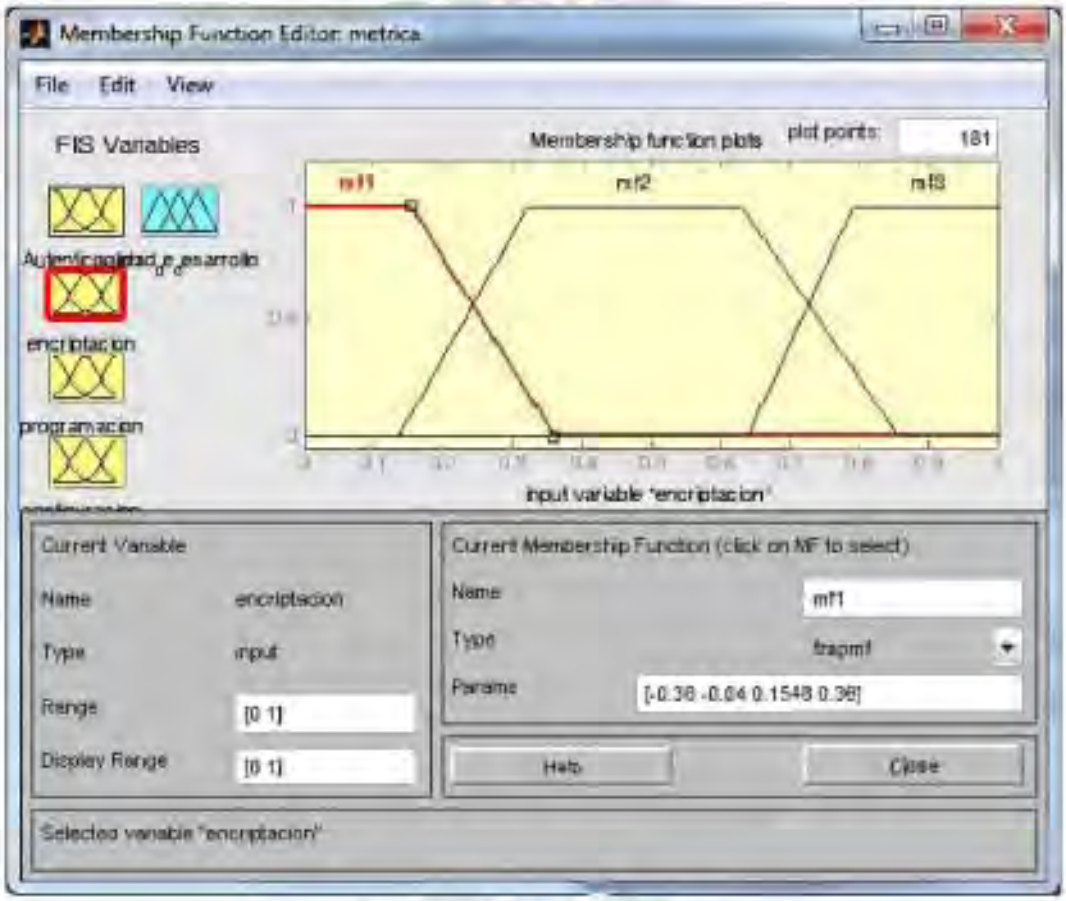

Figura 3.7 Variable encriptación
Fuente Elaboración Propia

VALOR LINGUISTICO	NOMBRE DEL CONJUNTO	FUNCION CARACTERISTICA	DEFINICION
1	BAJA	TRAPMF	$\mu(x) = \begin{cases} 0; 0 > x \\ 1; 0 \leq x \leq 0{,}15 \\ \frac{0{,}35 - x}{0{,}2}; 0{,}15 < x \leq 0{,}35 \\ 0; 0{,}35 < x \end{cases}$
2	MEDIA	TRAPMF	$\mu(x) = \begin{cases} 0; x < 0{,}15 \\ \frac{0{,}15 - x}{0{,}2}; 0{,}15 \leq x < 0{,}35 \\ 1; 0{,}35 \leq x \leq 0{,}65 \\ \frac{0{,}65 - x}{0{,}2}; 0{,}65 < x \leq 0{,}85 \\ 0; 0{,}85 < x \end{cases}$
3	ALTA	TRAPMF	$\mu(x) = \begin{cases} 0; 0{,}65 > x \\ \frac{0{,}65 - x}{0{,}1}; 0{,}65 \leq x \leq 0{,}75 \\ 1; 0{,}75 < x \leq 1 \\ 0; 1 < x \end{cases}$

Tabla 3.7: Función encriptación
Fuente: Elaboración Propia

Variables de entrada:	Programación
Función característica:	Trapezoidal
Universo de discurso	[0..1]
Tipo de Variable:	Dependiente
Valores lingüísticos:	3

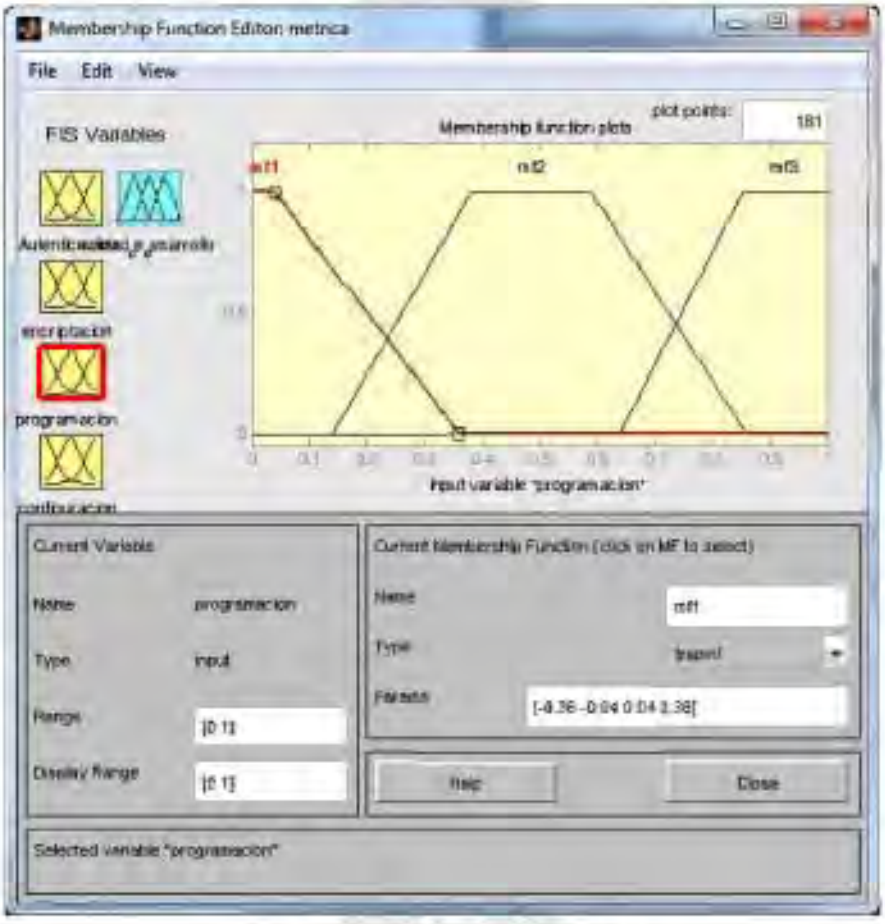

Figura 3.8 Variable programación
Fuente Elaboración Propia

VALOR LINGUISTICO	NOMBRE DEL CONJUNTO	FUNCION CARACTERISTICA	DEFINICION
1	BAJA	TRAPMF	$\mu(x) = \begin{cases} 0; 0 > x \\ \square \\ 1; 0 \leq x \leq 0{,}2 \\ \square \\ \frac{0{,}35 - x}{0{,}15}; 0{,}2 < x \leq 0{,}35 \\ \square \\ 0; 0{,}35 < x \end{cases}$
2	MEDIA	TRAPMF	$\mu(x) = \begin{cases} 0; 0 \leq x < 0.15 \\ \square \\ \frac{0.35 - x}{0.2}; 0.15 \leq x < 0.35 \\ \square \\ 1; 0.35 \leq x \leq 0.65 \\ \square \\ \frac{0.65 - x}{0.2}; 0.65 < x \leq 0.85 \\ \square \\ 0; 0.65 < x \end{cases}$
3	ALTA	TRAPMF	$\mu(x) = \begin{cases} 0; 0.65 > x \\ \square \\ \frac{0.65 - x}{0.1}; 0.65 \leq x \leq 0.75 \\ \square \\ 1; 0.75 < x \leq 1 \\ \square \\ 0; 1 < x \end{cases}$

Tabla 3.8: Función programación
Fuente: Elaboración Propia

Variables de entrada:	Configuración
Función característica:	Trapezoidal
Universo de discurso	[0..1]
Tipo de Variable:	Dependiente
Valores lingüísticos:	3

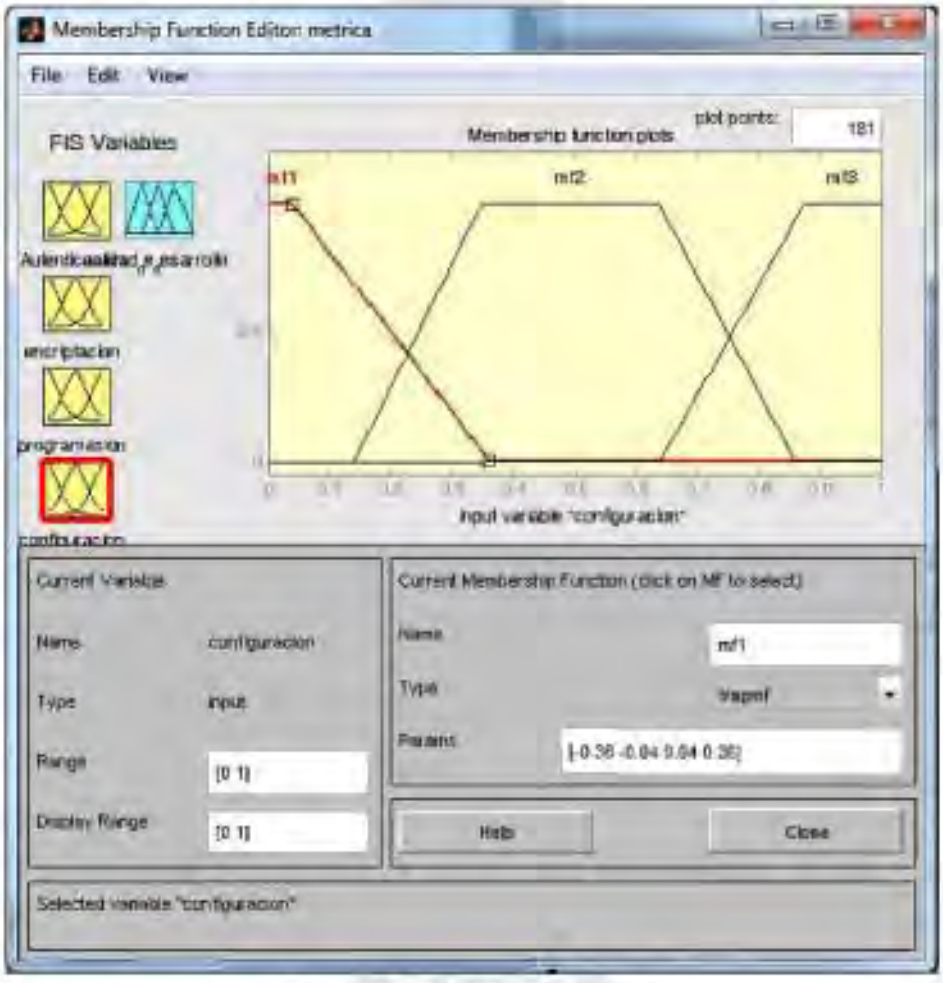

Figura 3.9 Variable configuración
Fuente Elaboración Propia

VALOR LINGUISTICO	NOMBRE DEL CONJUNTO	FUNCION CARACTERISTICA	DEFINICION
1	BAJA	TRAPMF	$\mu(x) = \begin{cases} 0; 0 > x \\ \square \\ 1; 0 \leq x \leq 0{,}15 \\ \square \\ \frac{0{,}35 - x}{0{,}2}; 0{,}15 < x \leq 0{,}35 \\ \square \\ 0; 0{,}35 < x \end{cases}$
2	MEDIA	TRAPMF	$\mu(x) = \begin{cases} 0; x < 0.15 \\ \square \\ \frac{0.15 - x}{0.2}; 0.15 \leq x < 0.35 \\ \square \\ 1; 0.35 \leq x \leq 0.65 \\ \square \\ \frac{0.65 - x}{0.2}; 0.65 < x \leq 0.85 \\ \square \\ 0; 0.85 < x \end{cases}$
3	ALTA	TRAPMF	$\mu(x) = \begin{cases} 0; 0{,}65 > x \\ \square \\ \frac{0.65 - x}{0.1}; 0.65 \leq x \leq 0.75 \\ \square \\ 1; 0.75 < x \leq 1 \\ \square \\ 0; 1 < x \end{cases}$

Tabla 3.9: Función configuración
Fuente: Elaboración Propia

Variables de salida:	Calidad_de_desarrollo
Función característica:	Trapezoidal
Universo de discurso	[0..1]
Tipo de Variable:	Independiente
Valores lingüísticos:	3

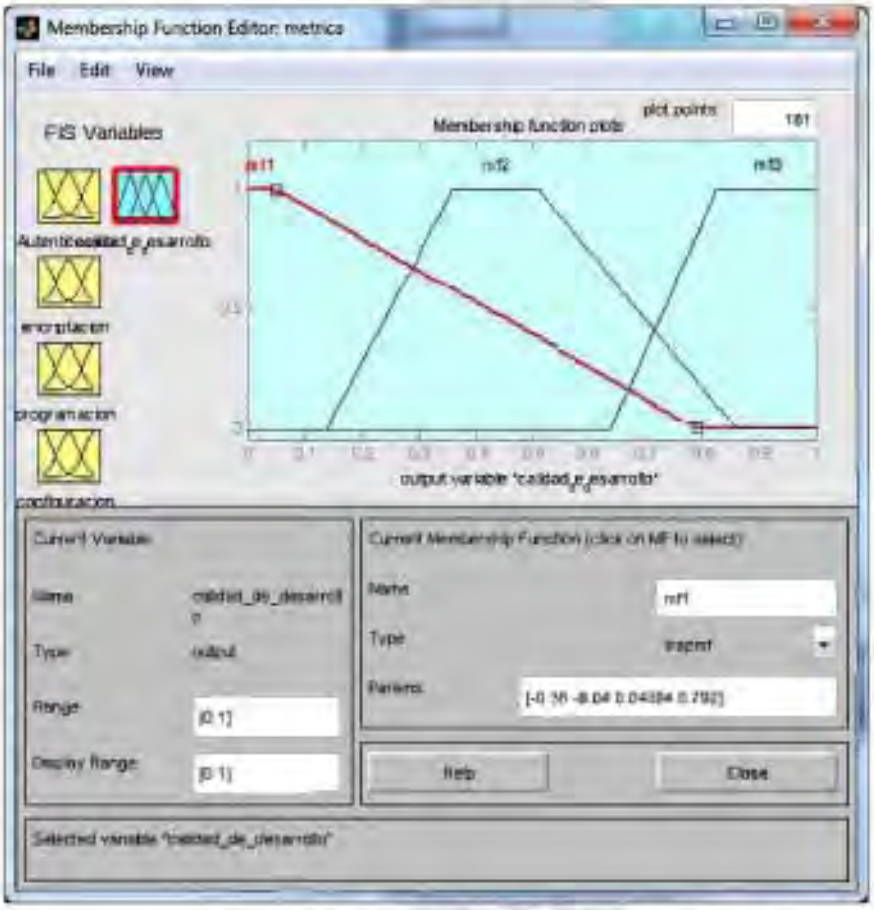

Figura 3.8 Variable de salida calidad de Desarrollo
Fuente Elaboración Propia

VALOR LINGUISTICO	NOMBRE DEL CONJUNTO	FUNCION CARACTERISTICA	DEFINICION
1	BAJA	TRAPMF	$\mu(x) = \begin{cases} 0; 0 > x \\ \square \\ 1; 0 \leq x \leq 0.05 \\ \square \\ \frac{0,8 - x}{0,75}; 0,05 < x \leq 0,8 \\ \square \\ 0; 0,8 < x \end{cases}$
2	MEDIA	TRAPMF	$\mu(x) = \begin{cases} 0; x < 0.15 \\ \square \\ \frac{0.15 - x}{0.2}; 0.15 \leq x < 0.35 \\ \square \\ 1; 0.35 \leq x \leq 0.55 \\ \square \\ \frac{0.85 - x}{0.3}; 0.55 < x \leq 0.85 \\ \square \\ 0; 0.85 < x \end{cases}$
3	ALTA	TRAPMF	$\mu(x) = \begin{cases} 0; 0.65 > x \\ \square \\ \frac{0.65 - x}{0.2}; 0.65 \leq x \leq 0.85 \\ \square \\ 1; 0.85 < x \leq 1 \\ \square \\ 0; 1 < x \end{cases}$

Tabla 3.10: Función calidad de desarrollo
Fuente: Elaboración Propia

3.10 ESTUDIO DE CASOS

Para respaldar las métricas propuestas, aplicamos las mismas a dos sistemas de información, ya que se contaba con la documentación correspondiente.

Figura 3.11: Casos de estudio
Fuente: Elaboración Propia

El sistema OMEGA SYSTEM, es un sistema de gestión de activos y personal parroquial, el SISTEMA HOTEL LP COLUMBUS, es el sistema de administración del hotel COLUMBUS; de acuerdo al documento de diseño de ambos sistemas obtuvimos los siguientes datos:

	OMEGA SYSTEM	SISTEMA HOTEL COLUMBUS
Control # de intentos	no	no
Uso de contraseñas	si	si
longitud de contraseña	No especif.	No especif.
contraseña numérica	si	no
contraseña alfanumérica	no	si
encriptación de datos	no	si
# de algoritmos de encriptación	0	1
# de validaciones de usuario	1	1
# de validaciones encriptadas	0	1
# total de consultas BD	11	23
# consultas BD protegidas	0	13
# total de campos de entrada	38	35
# validaciones de datos de entrada	2	16

Tabla 3.11. Tabla de especificaciones
Fuente: Elaboración Propia

De acuerdo a los datos de la anterior tabla, generamos los resultados mostrados en la tabla 3.10 con las funciones de criterio elemental definidas en la tabla 3.3.

CAMPO DE ACCION	FUNCIONES ASOCIADAS	OMEGA SYSTEM	HOTEL LP COLUMBUS
Autenticación	$u_0 = \sum_{i=1}^{n} w_i x_i / \sum_{i=1}^{n} w_i$ w_i = peso mecanismo de protección. número de incentivos $x_1 = \frac{\#\ validaciones\ correctas}{\#\ validaciones\ de\ usuario}$ uso de contraseñas $x_2 = \sum_{i=1}^{n} w_i A_i / \sum_{i=1}^{n} w_i$ A_i = peso según contraseña.	$u_0 = 0.266$ número de intentos $x_1 = 0$ uso de contraseñas $x_2 = 0.18$	$u_0 = 0.38$ número de intentos $x_1 = 0$ uso de contraseñas $x_2 = 0.14$
Encriptación	$x_1 = \sum_{i=0}^{n} x_i / n$ donde: $x_i = \frac{\#\ validaciones\ encriptadas}{\#\ validaciones\ de\ usuario}$ x_2 = peso según número de alg.	$u_1 = 0$ donde: $x_1 = 0$ $x_2 = 0$	$u_1 = 0.05$ donde: $x_1 = 1$ $x_2 = 0.7$
Programación	$u_2 = \frac{\#\ consultas\ protegidas}{\#\ conexiones\ a\ la\ BD}$	$u_2 = 0$	$u_2 = 0.56$
Abuso de funcionalidad	$u_3 = \frac{\#\ campos\ de\ entrada\ prot.}{\#\ campos\ de\ entrada}$	$u_3 = 0.052$	$u_3 = 0.46$
TOTAL	$Indicador\ elemental = \sum_{i=1}^{n} w_i x_i / \sum_{i=1}^{n} w_i$	$IE = 0.06$	$IE = 0.59$

Tabla 3.12. Aplicación de las funciones de criterio elemental

Fuente: Elaboración propia

De los resultados anteriores, intuitivamente podemos asumir que el segundo sistema es más seguro que el primero, pero esto daría lugar a ambigüedades, por lo que ahora los valores obtenidos de cada grupo serán evaluados con lógica difusa y así realizaremos una comparación entre el indicador elemental hallado y el resultado de este modulo.

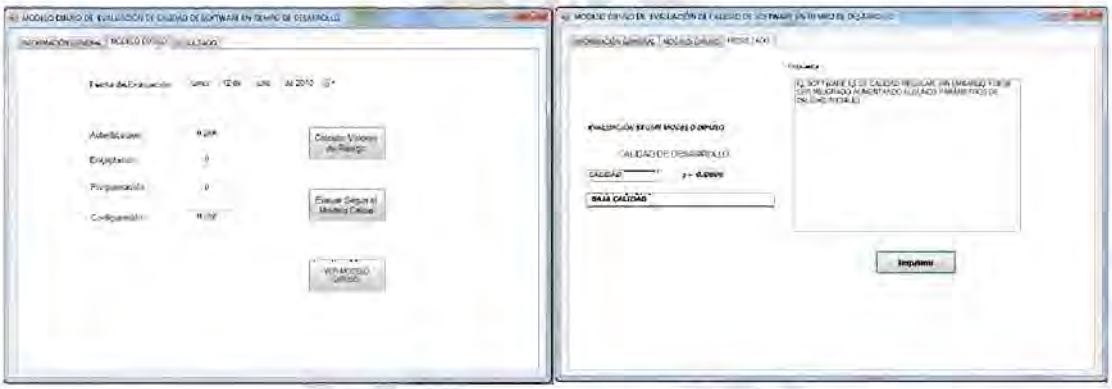

Figura 3.12: Evaluación OMEGA SYSTEM
Fuente: elaboración propia

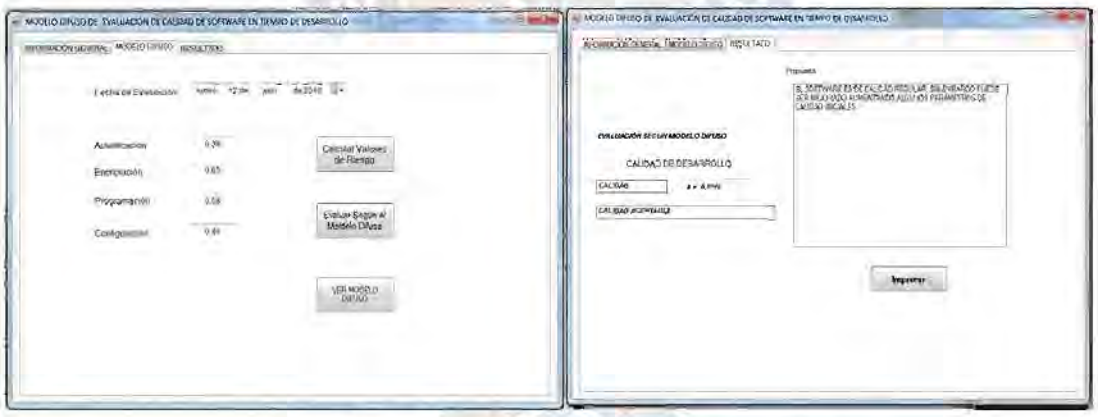

Figura 3.13: Evaluación HOTEL LP COLUMBUS
Fuente: elaboración propia

Comparando ambos resultados, podemos respaldar nuestra medida y su interpretación sobre la seguridad que poseen los sistemas que fueron nuestros casos de estudio. Por lo que podemos aceptar nuestra métrica como válida, ya que además posee los atributos que describen a una métrica.

CONCLUSIONES

4.1 CONCLUSIONES Y RECOMENDACIONES

El avance en el desarrollo de sistemas de software está en constante crecimiento, y la complejidad añadida a cada uno de estos junto con la funcionalidad y las técnicas que se deben considerar para asegurar la calidad de los mismos no es tomada en cuenta con la debida importancia.

A pesar de este crecimiento en la demanda y complejidad, la elaboración de métricas que permitan asegurar la calidad del software en la fase de desarrollo no se han logrado desarrollar de manera satisfactoria, no están acompañando este rápido proceso.

Más bien las prácticas actuales para desarrollar software son ad hoc, y el aseguramiento y el control de la calidad son, por lo general, procesos poco tenidos en cuenta. Esto motiva y urge a un cambio de paradigma en la prosecución de estos procesos y prácticas para minimizar la crisis en los desarrollos de software y por lo tanto propender a la calidad y a una efectiva evolución de las fases de desarrollo.

En este contexto, la tesis *Elaboración de métricas de calidad para evaluar el software durante su desarrollo* realiza un aporte ingenieril al sentar las bases y construir las estructuras para el uso de un enfoque sistemático, disciplinado y cuantitativo que se adecue a la evaluación, comparación y análisis de la calidad de aplicaciones más o menos complejas.

Discutimos las fases y actividades principales y los constructores de proceso para realizarlas en el capítulo 2 y 3.

Durante la fase de demostración, se considera un conjunto de atributos que son tomados en cuenta como un factor principal a ser considerados para lograr una efectiva evaluación del tipo de desarrollo que se realiza.

Posteriormente se establece criterios de evaluación de calidad elemental para cada atributo cuantificable, escalas, valores y rangos críticos, funciones para determinar la preferencia elemental, entre otros asuntos.

Una vez definidos y consensuados los criterios para medir cada atributo, se ejecuta el proceso de recolección de datos, el de cómputo de las variables y las preferencias elementales, y la documentación de los resultados. Los valores elementales obtenidos sirven de entrada al proceso de agregación.

Durante la fase de Definición e Implementación de la métrica, se discute dos enfoques cuantitativos de utilidad en las actividades de estructuración de atributos y cálculo de puntaje para la calidad. Específicamente, presentamos fortalezas y debilidades de los modelos de puntaje aditivos y lineales con respecto al modelo de agregación. Este fue el modelo seleccionado debido a la cantidad de atributos, sub-características y características intervinientes en los casos de estudio realizados.

Una vez que fueron estructurados y acordados todos los criterios, pesos y operadores de agregación, los evaluadores ejecutan una de dos posibles herramientas para calcular los resultados parciales y finales: uno basado en medidas de tendencia central como es la media geométrica y la otra es la que está basada en un grupo de conjuntos difusos, los que a su vez evalúan las características de cada atributo obtenido y genera un resultado que se aproxima al resultado obtenido por el método anterior.

En resumen se llegan a las siguientes conclusiones:

Se identificaron un conjunto de variables que influyen en la calidad del desarrollo del software cumpliéndose así los objetivos trazados al inicio de este estudio.

Se cumplió con la formulación de una nueva métrica para asegurar la calidad del software en tiempo de desarrollo, ha sido plenamente elaborada, llegando a determinar que las variables consideradas son muy importantes para establecer cómo influyen en la calidad del software.

Se estableció que este campo de estudio aún es muy poco estudiada, ya que generalmente tomar una evaluación al software en desarrollo no es considerada como una etapa importante y más al contrario existen herramientas de control de calidad del software solo cuando el producto está concluido y no así cuando está en desarrollo.

Algunas limitaciones encontradas en este estudio determinan que la evaluación de la calidad es una herramienta más que todo subjetiva por lo que se podría en un futuro desarrollar herramientas basadas en inteligencia artificial de tal manera que la evaluación pase del entorno subjetivo al entorno objetivo y cuantificable.

Así mismo una línea de investigación en el área de la ingeniería de software está relacionada con la formulación de nuevas métricas más asequibles a la realidad boliviana, es decir, se deben establecer teorías y herramientas locales adaptadas a nuestro entorno.

REFERENCIAS

5.1 BIBLIOGRAFIA

- Choque, G (2002), "Ingeniería del software: Principios y conceptos". Publicado en Diciembre de 2002
- Ejiogu L. (1991), "Software engineering whit formal metrics" Publishing QED, 1991
- IEEE (1993), "IEEE standard Glossary of software engineering terminology", edición 1993
- Lethbridge T. (2003), "Comunicación privada sobre métricas de software", Ed. Methuen 2003
- Lord Kelvin (2004), "Epónimos científicos" visitado en el sitio: http://www.uch.ceu.es/principal/eponimos_cientificos/eponimos/kelvin.pdf
- Minguet (2003), "La calidad del software y su medida", Ed. Ramon Areces 2003
- Nick (2001), "La segundad en la ingeniería del software", visitado en el sitio: http://www.acm.org/crossroads/espanol/xrds7-4/ompatrol74.html
- Pressman, Roger (2005), "Ingeniería del software. Un enfoque práctico", Ed. McGraw-Hill 2005
- Roche J. M. (1994), "Principios de medidas y métricas del software" en Software Engineering Notes, ACM, vol 19, Nro. 1, enero de 1994
- Saray (mayo de 2009), A. J. "Historia del software", visitado en el sitio: http://www.mitecnologico.com/Main/HistoriaIngenieriaSoftware, en fecha 31 de mayo de 2009
- Williams, S, "Teoría unificada de la evolución del software", visitado en el sitio: http://www.salon.com/tech/feature/2002/04/08/lehman/index.html

ANEXOS

ENCUESTA: IMPACTO DE VULNERABILIDADES

****

1. Indique cual de los siguientes campos de acción, representa un campo de acción de alto riesgo

   a. Autenticación ___
   b. Encriptación ___
   c. Programación-Inyección de código malicioso ___
   d. Funcionalidad-Manejo de excepciones ___

2. *Autenticación:* En qué porcentaje* considera el nivel de seguridad que ofrecen las siguientes características:

   a. Control de numero de intentos de ingreso al sistema ___
   b. Uso de contraseñas ___

   *La suma de ambos no debe exceder el 100%

3. *Autenticación:* En una escala del 1 al 10, indique el nivel de seguridad que ofrece la longitud las contraseñas y la naturaleza de la misma.

   a. 1 – 8 Caracteres ___
   b. 9 – 13 Caracteres ___
   c. 14 ó más caracteres ___

4. *Autenticación:* En una escala del 1 al 10, indique el nivel de seguridad que ofrece la naturaleza de las contraseñas.

   a. Caracteres solo numéricos ___
   b. Caracteres alfanuméricos ___

5. *Autenticación*: En qué porcentaje* considera el nivel de seguridad que ofrecen las siguientes características:

   a. Longitud de la contraseña ___
   b. Naturaleza de las contraseñas ___

   *La suma de ambos no debe exceder el 100%

6. *Encriptación*: En una escala del 1 al 10, indique el nivel de seguridad que ofrece el número de algoritmos de encriptación utilizados.

   a. 1 Algoritmo ___
   b. 2 ó más algoritmos ___

RESULTADOS DE LA ENCUESTA

1. Indique cual de los siguientes campos de acción, representa un campo de acción de alto riesgo.

2. En qué porcentaje* considera el nivel de seguridad que ofrecen las siguientes caracteristicas:

   a. Control de numero de intentos de ingreso al sistema __
   b. Uso de contraseñas __

*La suma de ambos no debe exceder el 100%

3. *Autenticación:* En una escala del 1 al 10, indique el nivel de seguridad que ofrece la longitud las contraseñas

   a. 1 – 8 Caracteres **1**
   b. 9 – 13 Caracteres **6**
   c. 14 ó más caracteres **9**

4. *Autenticación:* En una escala del 1 al 10, indique el nivel de seguridad que ofrece la naturaleza de las contraseñas.

   a. Caracteres solo numéricos **5**
   b. Caracteres alfanuméricos **9**

5. *Autenticación:* En qué porcentaje* considera el nivel de seguridad que ofrecen las siguientes características:

   c. Longitud de la contraseña ___
   d. Naturaleza de las contraseñas ___

   *La suma de ambos no debe exceder el 100%

6. *Encriptación.* En una escala del 1 al 10, indique el nivel de seguridad que ofrece el número de algoritmos de encriptación utilizados.

e. 1 Algoritmo	**7**
f. 2 ó más algoritmos	**9**

Nota.-

Los resultados mostrados de las preguntas 3, 4 y 6, fueron obtenidos aplicando la media aritmética del total de las posibles respuestas.

METRICAS DE CALIDAD

1. Sistema de gestión de activos y personal

Parámetros de medición punto función:

	COMPLEJIDAD				
Tipo de función	BAJA	MEDIA	ALTA	VALOR	APORTE
Entradas externas	3	4	6	22	88
Salidas externas	4	5	7	13	65
Consultas externas	3	4	6	13	52
Archivos logicos internos	7	10	15	25	250
Archivos interfaz externa	5	7	10	0	0
				TOTAL	455

UFP = 455

Para calcular el punto función ajustado PF = UFP x AF

Valores de ajuste:

Factores de influencia en la Dificultad del Sistema	Grado	Valor Max
1. Comunicaciones de datos	0	5
2. Procesamiento distribuido	0	5
3. Objetivos de rendimiento	3	5
4. configuración de uso intensivo	4	5
5. Tasas de transacción rápidas	2	5
6. Entrada de datos en linea	0	5
7. Amigabilidad en el diseño	4	5
8. Actualización de datos en linea	0	5
9. Procesamiento complejo	2	5
10. Reusabilidad	4	5
11. Facilidad de instalación	4	5
12. Facilidad operacional	4	5
13. Adaptabilidad	3	5
14. Versatilidad	4	5
Total Fi	**34**	**70**

Total Fi = 34

AF = Total Fi * 0.01 + 0.65 = 0.99

PF = UFP x AF = 455 x 0.99 = 450.45

El punto función obtenido supone 450 aproximadamente, referente al tamaño del sistema. Haciendo un análisis con la escala del punto función mostrada en la siguiente tabla:

Escala	Observación
PF > 300	Optima
200 > PF < 300	Buena
100 > PF < 200	Suficiente
PF > 100	Deficiente

Se concluye que el sistema tiene una funcionalidad óptima.

Calculando el ajuste tenemos:

El valor promedio de Fi (calculado) es = 34

El valor máximo de Fi es = 70

La cuenta total del punto función sin ajustar es = 455

Reemplazando estos valores en la ecuación PF tenemos:

PF (máximo) = 455 x [(0.01 * 70) + 0.65] = 614.25

Entonces, sacando el promedio de estos dos resultados obtenemos:

Promedio = 450 / 614 = 0.73

De donde:

Funcionalidad del sistema = 0,73 * 100 = 73%

1.2FACTORES DE CALIDAD DE MC CALL

Revisión del producto

- Facilidad de mantenimiento: f1 = 9
- Flexibilidad: f2 = 8
- Facilidad de prueba : f3 = 7

Entonces: Fq = 1/3(f1 + f2 + f3) = 8

Transición del producto

- Portabilidad: p = 7
- Reusabilidad: r = 6
- Interoperabilidad: i = 5

De donde: Fq = 1/3(p + r + i) = 6

Operación del producto

- Corrección: c = 10
- Fiabilidad: f = 9
- Usabilidad: u = 8
- Integridad: i = 9
- Eficiencia: e = 9

De donde: Fq = 1/5(c + f + u + i + e) = 7

2. Sistema de administración HOTEL COLUMBUS

Parámetros de medición punto función:

	COMPLEJIDAD				
Tipo de función	BAJA	MEDIA	ALTA	VALOR	APORTE
Entradas externas	3	***4***	6	28	112
Salidas externas	4	***5***	7	16	80
Consultas externas	3	***4***	6	13	52
Archivos lógicos Internos	7	***10***	15	20	200
Archivos interfaz externa	5	***7***	10	0	0
				TOTAL	444

UFP = 444

Para calcular el punto función ajustado PF = UFP x AF

Valores de ajuste:

Factores de influencia en la Dificultad del Sistema	Grado	Valor Max.
1. Comunicaciones de datos	0	5
2. Procesamiento distribuido	0	5
3. Objetivos de rendimiento	4	5
4. configuración de uso intensivo	4	5
5. Tasas de transacción rápidas	1	5
6. Entrada de datos en línea	0	5
7. Amigabilidad en el diseño	4	5
8. Actualización de datos en línea	2	5
9. Procesamiento complejo	2	5
10. Reusabilidad	4	5
11. Facilidad de instalación	4	5
12. Facilidad operacional	4	5
13. Adaptabilidad	3	5
14. Versatilidad	4	5
Total Fi	**36**	**70**

Total Fi = 36

AF = Total Fi * 0.01 + 0.65 = 1.01

PF = UFP x AF = 444 x 1.01 = 448.44

El punto función obtenido supone 448 aproximadamente, referente al tamaño del sistema. Haciendo un análisis con la escala del punto función mostrada en la siguiente tabla:

Escala	Observación
PF > 300	Optima
200 > PF < 300	Buena
100 > PF < 200	Suficiente
PF > 100	Deficiente

Se concluye que el sistema tiene una funcionalidad óptima.

Calculando el ajuste tenemos:

El valor promedio de Fi (calculado) es = 36

El valor máximo de Fi es = 70

La cuenta total del punto función sin ajustar es = 444

Reemplazando estos valores en la ecuación PF tenemos:

PF (máximo) = 444 x [(0,01 * 70) + 0.65] = 599.4

Entonces, sacando el promedio de estos dos resultados obtenemos:

Promedio = 450 / 614 = 0.74

De donde:

Funcionalidad del sistema = 0,74 * 100 = 74%

2.2 FACTORES DE CALIDAD DE MC CALL

Revisión del producto

- Facilidad de mantenimiento: f1 = 9
- Flexibilidad: f2 = 9
- Facilidad de prueba : f3 = 8

Entonces: Fq = 1/3(f1 + f2 + f3) = 8.6

Transición del producto

- Portabilidad: p = 8
- Reusabilidad: r = 6
- Interoperabilidad: i = 6

De donde: Fq = 1/3(p + r + i) = 7.3

Operación del producto

- Corrección: c = 10
- Fiabilidad: f = 9
- Usabilidad: u = 8
- Integridad: i = 9
- Eficiencia: e = 9

De donde: Fq = 1/5(c + f + u + i + e) = 7